AN INVESTIGATION OF PRODUCTION SCHEDULING PROBLEMS MOTIVATED BY SEMICONDUCTOR MANUFACTURING

By

JEFFREY WILLIAM HERRMANN

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1993

ACKNOWLEDGMENTS

Years of learning are not sufficient for one who wishes to prepare a dissertation; also necessary is an environment which encourages one to develop the talents which God has given and to become an intelligent person, a responsible person, a kind person, a successful person. For providing such a setting I thank my family.

To many others am I indebted: to the chairman of my supervisory committee, Dr. Chung-Yee Lee, who has been an exceptional influence on my development as a researcher, teacher, and scholar; to department chairman Dr. Jack Elzinga, Dr. Boghos Sivazlian, and the members of my committee; to all who were involved in the Harris project and whose effort was critical to this work; and to the quality friends, classmates, and teachers whom I have known and who have all added something valuable to my life.

DISCLAIMER

This material is based upon work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

page

CHAPTERS

Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

AN INVESTIGATION OF PRODUCTION SCHEDULING PROBLEMS MOTIVATED BY SEMICONDUCTOR MANUFACTURING

By

Jeffrey William Herrmann

December, 1993

Chairman: Dr. Chung-Yee Lee
Major Department: Industrial and Systems Engineering

Manufacturing and service organizations frequently face the challenges of making high-quality products quickly and of delivering those products to their customers on-time. Improvements in the scheduling of their operations can often contribute to their success in meeting these goals. However, as manufacturing processes become more complex, the difficulty of finding good production schedules increases.

This dissertation addresses dynamic deterministic job shop scheduling, a problem that occurs in many manufacturing environments. The problem is among the most difficult scheduling problems, and few solution procedures have been implemented. The approach in this research is to consider specific subproblems that are motivated by semiconductor test operations and to develop genetic algorithms that exploit alternative search spaces.

The research includes new analytical and empirical results for previously unstudied one-machine class scheduling problems and three-machine look-ahead problems. The one-machine problems include sequence-dependent setup times. In the three-machine problems, two groups of jobs are processed on separate second-stage machines. Testing shows that a new type of genetic algorithm can find good schedules for the one-machine problems by adjusting the problem data while using an appropriate heuristic. An approximation algorithm for the three-machine problem is able to find near-optimal schedules.

Moreover, this dissertation describes the development and application of a global job shop scheduling system for the semiconductor test area. This system uses a detailed deterministic simulation model of the shop floor, data about the current status of the shop, and a genetic algorithm to search over combinations of dispatching rules in order to create a good shift schedule. These rules include those motivated by this research into the one-machine and three-machine problems. The scheduling system is able to adapt to changing conditions each shift.

The benefits of this work consist of the identification of dominance properties for the one-machine class scheduling and three-machine look-ahead problems, the development of a problem space genetic algorithm, the definition of new look-ahead heuristics, the creation of a new genetic algorithm for global scheduling, and the implementation of these results to the actual semiconductor test floor that is the motivation for this work.

CHAPTER 1
INTRODUCTION

Why investigate production scheduling problems motivated by semiconductor manufacturing? Because scheduling works. Scheduling, the term, refers to the process of assigning tasks to resources and determining when each task will be done. Scheduling is not, however, limited to manufacturing, since this type of problem occurs in other activities. Scheduling, the science, embodies knowledge about models, techniques, and insights related to actual systems (Baker, 1992). This field is an important part of operations research. In order to compete effectively in the marketplace, firms have used operations research as a tool for understanding and improving their manufacturing systems.

This is especially true in the manufacturing of high-technology products, where the new, complicated processes lead to production scheduling problems that are not well-solved. The post-assembly testing of semiconductors, for instance, is a complicated job shop scheduling problem; among the difficulties is the presence of sequence-dependent setups at certain operations. Moreover, scheduling, which has been studied from the earliest days of operations research, has received new attention due to the successful implementation of just-in-time systems that emphasize the close coordination of resources and the maintenance of low work-in-process inventory levels.

Although the research in production scheduling has yielded a large body of knowledge which has been successfully applied to a number of areas, there still remain problems to be solved. The problems previously studied have always simplified reality in some way. Adding realism creates problems that resist simple solution techniques. Further research into these types of problems is necessary in order to increase our ability to control manufacturing processes effectively.

The research described in this dissertation falls into this category. It includes factors that model reality more closely and studies a number of different problems in an effort to gain insight into how improved job shop scheduling may be achieved. This insight will be applied to the semiconductor test area being studied.

The problems that this research investigates are motivated by a high-technology field where much time has been spent investigating manufacturing systems, namely the semiconductor industry. The research is particularly motivated by the operations of semiconductor testing, although the problems can also be found in the processes of other manufacturing industries.

This introduction includes a number of sections about topics that will arise in the scheduling of semiconductor test operations. These topics are introduced here in order to describe how they are related to this research. The first of these is an overview of semiconductor manufacturing.

1.1 Semiconductor Manufacturing

In order to give some context to the problems being studied, this section describes the general flow of semiconductor manufacturing.

The manufacturing of semiconductors consists of many complex steps. The first process is that of wafer fabrication, done in a super-clean environment in order to protect the delicate structures on the wafers from contaminants. The wafers of silicon undergo repeated applications of the cycle of photolithography, etching, and diffusion in order to build, layer by layer, the different electronic devices. A wafer will contain a number of identical circuits laid out in a grid pattern on the wafer. In the next process, probe, these individual circuits are inspected with a wafer probe and marked if defective. In the assembly process, another clean area, the wafers are cut into the separate circuits. Good circuits are placed into the packages that protect them from the environment and that provide electrical connections to allow them to interact with the world. The identical devices from a lot of identical wafers become a lot that then moves into the test

facility, an area where environmental safeguards protect the devices from static electricity that could still damage the circuits.

The test process consists of initial testing, burn-in, and final testing. The testing includes electrical testing of each device and other testing of the device packages. Burn-in consists of subjecting the semiconductor chips to extreme thermal stresses. Other non-test operations, such as serialization and brand, take place during the process.

Not all of the lots follow the same route through the test floor. Different products are tested on different machines and undergo a different set of tests. Thus, the test area is a job shop, and the scheduling problem is a job shop scheduling problem. This job shop of semiconductor test operations has sequence-dependent setup times, re-entrant flows, batch processing, routes that temporarily leave the test area, and other complications that lead to many interesting scheduling questions.

The primary management objectives for this area are customer satisfaction and making profit. Because testing is the last process in the manufacturing flow, the output of this area affects how well the company can get orders to its customers on time and how much product the company can ship to generate revenue.

The particular semiconductor test area being studied has implemented a decision support system to assist in the scheduling of lots through the area. The programs in the system can order the lots waiting for an workstation with priority rules that include information about lateness, priority, and setups. As part of the facility-wide computer-integrated manufacturing system, it is able to access real-time information about the lots. The system is used to create schedules for a short time period (two to four hours), since dispatching more often would require excessive computer resources. Lots that are tardy and lots that have some externally-imposed priority are expedited whenever possible.

1.2 Job Shop Scheduling

The semiconductor test area is a job shop, and optimizing the scheduling of the area is a job shop scheduling problem. This section discusses job shop scheduling.

Job shop scheduling consists of those scheduling problems in which different jobs may follow different routes through the shop. A job consists of a number of operations or tasks, which must be processed in the given order. There exists a specified machine (or workstation) that can perform each operation. These problems are generally the hardest to solve optimally, since few properties of optimal schedules are known and the number of possible solutions explodes as the problems increase in size. However, in many cases, the job shop is a better model of reality than one-machine, parallel-machine, or flow shop problems.

Because of the complexity of job shop scheduling, algorithms to find the optimal solution (in a reasonable amount of time) for any arbitrary objective function do not exist. Recent research has shown that the *shifting bottleneck* heuristic is successful at finding good solutions for a simple job shop scheduling problem. Traditionally, however, researchers have studied and schedulers have used dispatching rules to order the jobs waiting for processing at a machine. When a machine becomes available, it chooses from among the jobs in its queue by using a dispatching rule. Common dispatching rules employ processing times and due dates in both simple rules and complex combinations. These dispatching rules are often extensions of the algorithms used to solve one-machine problems.

In any shop, there may exist a bottleneck machine whose throughput is a limiting factor on the capacity of the shop. If so, the improved scheduling of this one machine becomes an important objective.

1.3 Look-ahead and Look-behind Scheduling

This section addresses a weakness of the traditional dispatching rule approach to scheduling job shops and defines two types of rules that overcome this weakness.

Traditional dispatching rules are myopic; they are concerned only with the machine to be scheduled and the jobs waiting for that machine. Improved scheduling may be realized by using rules that can consider more information, in order to see that critical lots are arriving soon or that a certain machine has an excessive queue. Look-ahead and look-behind scheduling includes procedures that look around the shop for more information to use in making a scheduling decision. With this additional information, they can hopefully produce better decisions. Look-ahead models consider the machines where the jobs will be headed after this stage. Look-behind models consider the jobs that will be arriving at this machine soon.

The terms *look-ahead* and *look-behind* are used to designate scheduling procedures that do more than consider just the state of one machine. Both types of models look into the future (where jobs will go and what jobs will be arriving). The difference is what part of the future they consider.

1.4 Setups

One of the complications of scheduling the operations in semiconductor testing is the presence of different product types that require different configurations on the same machine. The task of configuring a machine in order to process a job is a *setup*.

If the setup for a job is independent of the job that was scheduled before it on the same machine, the setup time can be included in the processing requirements of that job. If the setup for a job is *sequence-dependent*, that is, the time of the setup depends upon the immediate predecessor job, the scheduling problem becomes quite difficult.

Because the problems with sequence-dependent setups are quite difficult, researchers have examined special cases of the general problem. The most common problem is the *class scheduling problem*, where the jobs to be processed are grouped into a number of job *classes*. There exists no sequence-dependent setup between jobs from the same class, although there does exist a special setup, the *class setup*, when a job from one class is processed after one from another class. There may also exist a class setup for the first job processed. These class setups

may be sequence-dependent in that they depend upon the class that was last processed. Class scheduling is still difficult, though: the class scheduling extensions of one-machine problems that are easy to solve are usually NP-complete problems.

An example of class setups occurs in the electrical testing of assembled semiconductor devices on a machine which can test a number of different types of semiconductors. If a machine is scheduled to test a lot consisting of devices that are different from the devices tested in the previous lot, various setup tasks are required. These tasks may include changing a handler and load board that can process only certain types of semiconductor packages or loading a new test program on the tester. However, if the new lot consists of devices that are the same as the previous type, none of this setup is required. This is a class scheduling problem.

1.5 Smart-and-lucky Searches

As mentioned before, algorithms to solve the job shop scheduling problem optimally in reasonable time do not exist. In addition to the shifting bottleneck heuristic and the use of dispatching rules, one way to find good solutions is to search for them. This section will discuss standard local searches and new, more sophisticated heuristic searches.

One method of finding good solutions to hard optimization problems has been local search. A local search begins with some initial solution and moves from an incumbent solution to a better neighboring solution, ending when no improvement can be found. At this point, the search has reached a local optimum. Two common local searches are *hillclimbing* and *steepest descent.* In a hillclimbing search, the neighbors are chosen at random, and the first better neighbor found is chosen as the new solution. In a steepest descent search, the entire neighborhood of the incumbent solution is searched and the neighbor that has the best performance improvement is selected. In order to overcome the fact that these searches converge only to local optima, new heuristic searches have been developed. These include *simulated annealing*, *tabu search*, and *genetic algorithms.*

These heuristic searches are complex searches that are *smart* enough to escape from most local optima and are *lucky* enough to generally find good solutions.

Simulated annealing (SA) is a variant of the hillclimbing algorithm. Simulated annealing is so called because the algorithm views optimization as a process analogous to physical annealing, the cooling of a system until it reaches a low-energy state. In the same way that a system may pass through higher-energy states during the cooling process, the simulated annealing search occasionally moves to worse neighbors before settling into a good solution.

Tabu search (TS) is a variant of steepest descent. Given an incumbent solution, a TS canvasses the neighborhood of this solution in order to find the best allowable move. If the search is at a local optimum, it is forced to move away, and the move that would return to the previous solution is temporarily prohibited (tabu) by adding it to the tabu list for a number of moves. In this way, TS may continue to move away from a local optimum and find an area of the space that leads to another local optimum. An aspiration level insures that the search will make a move that leads to a solution better than any yet found, even if tabu. Thus, a tabu search works because the tabu list forces the search to explore new areas of the solution space. The short-term aspect of the memory and the aspiration level allow, however, the search to get to a global optimum.

A genetic algorithm (GA) is a procedure that mimics the adaptation that nature uses to find an optimal state. In genetic algorithms, solutions are represented as strings (chromosomes) of alleles. An allele is a bit of information about the solution. The search performs operations on the population of solutions. These operations are 1) the evaluation of individual fitness, 2) the formation of a gene pool, and 3) the recombination and mutation of genes to form a new population. After a period of time, good strings dominate the population, providing a good solution.

When applied to scheduling problems, simulated annealing and tabu search easily search the complex solution spaces with simple types of moves, generally find good solutions fast, and are smart and lucky variations of standard searches such as hillclimbing and steepest descent.

Genetic algorithms are something completely different, work well by using good strings and implicit parallelism, and are harder to implement.

1.6 Problem Space and Heuristic Space

Typical problem-solving searches like those described in the above section have examined the solution space. A solution can be a vector of values or a sequence of objects. Applying a heuristic to a problem yields a point in the solution space. This implies that a search over all of the possible heuristics would find a solution that gives the optimal objective function value.

Moreover, applying a heuristic to another, similar problem also generates a point in the solution space which can be evaluated for the original problem using the objective function. Thus, a search over the new problem space provides a way to solve the problem.

An example of a heuristic space is the space of vectors of m dispatching rules for a n-job and m-machine job shop scheduling problem, where each dispatching rule is applied to a different machine. Searching over the vectors of dispatching rules is similar to machine learning. For a complicated one-machine problem, a sample problem space would be the space of n-element vectors, where each element corresponds to an alternative due date for a job. Sorting the jobs by these alternative due dates creates a new solution.

1.7 Objective Functions

The management of a manufacturing system is concerned with many different performance measures, including customer service, inventory holding costs, throughput, and machine utilization. This section describe how the objectives used in scheduling mirror these concerns.

Scheduling problems include many different objective functions that attempt to model the concerns of those who are running the system. Most objectives are functions of the completion times of the jobs to be scheduled. The *makespan* is the maximum completion time, which is when all of the jobs are finished, and is some measure of the throughput of the system. The *total flowtime* is the total time that the jobs spend in the system and is a measure of the work-in-

process inventory held while processing the jobs. Other objective functions include due dates for each job, where a job is tardy if it completes after its due date. These objective functions are concerned with customer satisfaction, and include *maximum lateness*, which is the maximum difference between a completion time and a due date, the *number of tardy jobs*, and the *total tardiness*. Another interesting objective function is *earliness*, which is a measure of the holding cost incurred by storing finished product until it is delivered at its due date.

While all of these objective functions (and many more) have been studied for one-machine problems, most analysis of flow shop and job shop scheduling has focused on the minimization of makespan. Since this objective is not appropriate in semiconductor manufacturing, this research is especially interested in solving job shop scheduling problems with other objectives.

1.8 Overview of Research

There exist different approaches to attacking the job shop scheduling problem. One can use the shifting bottleneck algorithm to minimize the makespan, use good dispatching rules in a dynamic environment, or search for good global schedules. This research is concerned with the last two methods (see Figure 1.1). This research investigates the use of smart-and-lucky searches over the new search spaces to find good solutions to the job shop scheduling problem. Also investigated are a number of interesting one-machine class scheduling problems and look-ahead models in order to devise techniques that can be used as good dispatching rules. Finally, this research studies how these techniques may be applied to the actual semiconductor test floor that is the motivation for this work.

The benefits of this work include three areas: 1. The addition of results to the body of knowledge about specific scheduling problems, including the difficult class scheduling problems and little-studied look-ahead models. 2. The construction of a robust genetic algorithm for one-machine class scheduling problems. 3. The development and implementation of a genetic algorithm for global job shop scheduling. Moreover, this work continues the investigation of scheduling semiconductor test operations that has only recently begun.

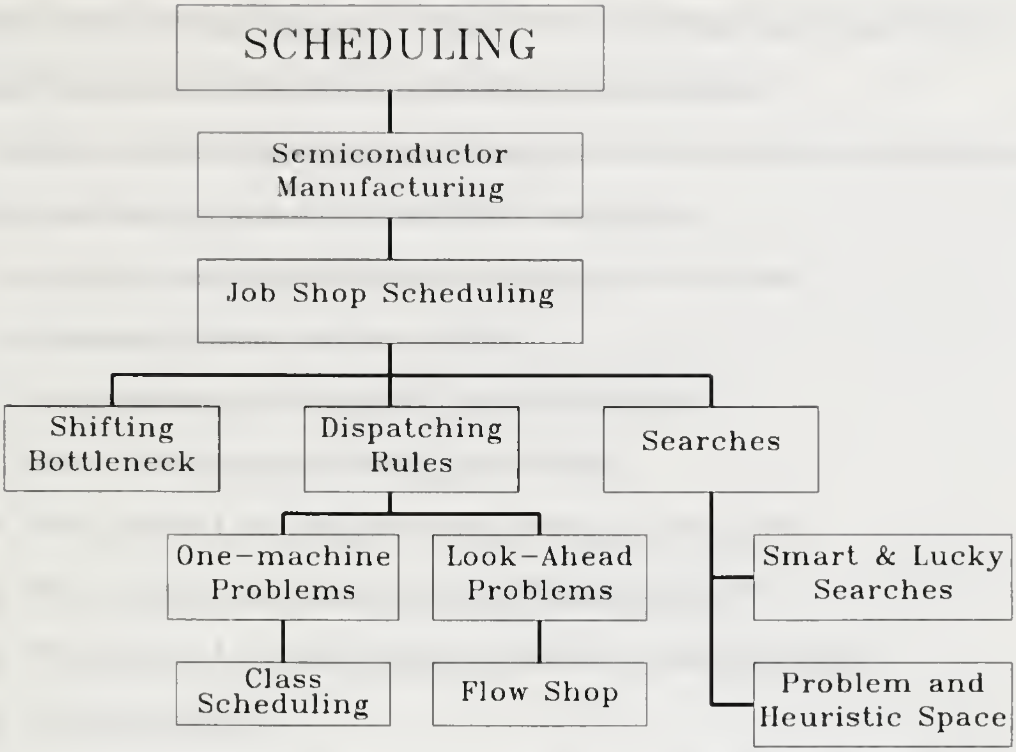


Figure 1.1. Relationship of dissertation topics.

1.9 Plan of Dissertation

This research investigates both one-machine and multi-machine problems. The one-machine problems studied are class scheduling problems that model the complicating factor of machine setups in the manufacturing process. This work also considers some three-machine general flow shop problems that are attempts to model how scheduling can be improved by considering the states of other machines. The research also investigates a general job shop scheduling problem in an attempt to see how new search spaces may be used for global job shop scheduling.

For these problems, the research has focused on finding good lower bounds, developing heuristics to find good solutions for the problems, and using genetic algorithms to find better solutions. Empirical testing of these bounds and heuristics serves as a way of evaluating the

[illegible]

[illegible]

[illegible]

heuristics and the searches. Also, good heuristics for the subproblems can be implemented as advanced dispatching rules for solving the job shop scheduling problem.

In addition, analytical results such as NP-completeness, lower bounds, error bounds, and optimality conditions are presented for the problems being studied.

The machine scheduling problems under investigation are as follows:

1. Constrained Flowtime with Setups (CFTS)
2. Class Scheduling with Release and Due Dates (CSRDD)
3. Flowtime with Setups and Release Dates (FTSRD)
4. Three-Machine Look-Ahead Scheduling: Makespan (3MLA-MS)
5. Three-Machine Look-Ahead Scheduling: Flowtime (3MLA-FT)
6. Three-Machine Look-Ahead Scheduling: Number of Tardy Jobs (3MNT)
7. Job shop scheduling

The first problem is the class scheduling extension of the one-machine problem of minimizing the total flowtime of a set of jobs that have deadlines. The second problem is the class scheduling problem where each job has a release date and a due date. The objective is to minimize the number of tardy jobs. The objective in the third problem is to minimize the total flowtime where each job has a release date.

The research next considers the three-machine look-ahead problems with the objectives of makespan, total flowtime, and number of tardy jobs.

Finally, this research investigates the general job shop scheduling problem and a heuristic space approach to finding good solutions with a genetic algorithm.

The dissertation consists of the following five chapters: the background; the research on one-machine class scheduling problems, on three-machine look-ahead problems, and on job shop scheduling; and the conclusions.

The background material contains detailed information about the problems and methods under investigation and summaries of a number of papers that are related to this work. The topics include semiconductor test operations and semiconductor scheduling, job shop scheduling

techniques, class scheduling, some one-machine problems, smart-and-lucky searches, problem and heuristic space, look-ahead and flow shop scheduling, burn-in scheduling, and NP-completeness.

The discussion of the research contains reports on the problems that have been investigated: three one-machine class scheduling problems, three three-machine general flow shop problems, and a heuristic search for use on the general job shop scheduling problem. In the conclusions we summarize this research and identify some directions for future research.

CHAPTER 2
BACKGROUND

This chapter of the dissertation performs two functions: it elaborates on the topics in the introduction and provides a review of the relevant literature. The discussion in this chapter is both wide-ranging and descriptive. In Chapters 3, 4, and 5, discussions of previous research will be problem-specific and less explicit.

The first section will describe in more detail the particular operations associated with the testing of semiconductor devices. Remaining sections will discuss papers on semiconductor manufacturing, job shop scheduling techniques, and flow shop scheduling problems. Also included in this chapter are sections on look-ahead and look-behind scheduling, the scheduling of the burn-in operation, class scheduling, some one-machine problems, smart-and-lucky searches, problem and heuristic space, and NP-completeness.

2.1 Semiconductor Test Operations

The manufacturing of semiconductors consists of many complex steps. As described in the introduction, this includes wafer fabrication, probe, assembly, and test. This research is concerned with this last facility. Although the routes of lots through the test process vary significantly over the many different types of products, general trends can be described. Figure 2.1 shows what these routes look like for seventeen representative products of a test area. The numbers on each arc are the number of products that follow that path.

Presented here is a description of the operations in a typical product route. The terms semiconductor and device describe the same thing.

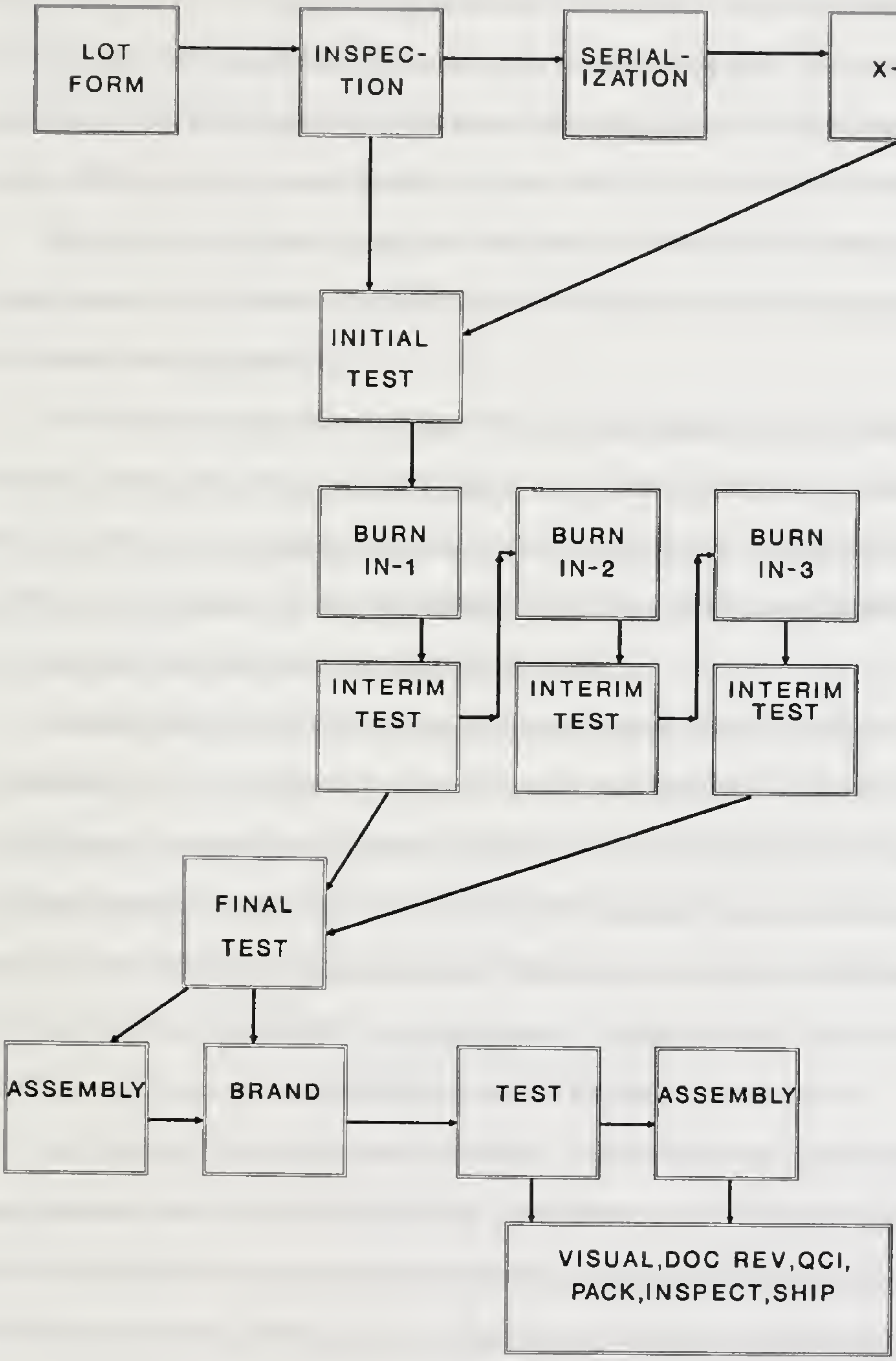


Figure 2.1. Product Routes in the Test Area.

[illegible]

The lot arrives at the lotform area with an assembly traveler that describes the history of the lot up to that point. A lot is started through the test floor when it is needed to meet back-ordered demand or when the area planner includes it on the weekly release plan. The test traveler for this product is printed, and engineering information is extracted from the database and attached to the traveler. The tubes of devices are placed into a box, and the box is moved to a waiting area.

The lot is then serialized. Done in the brand area, serialization is the process of giving each device a unique serial number. This process is labor-intensive as someone must stamp each device with a branding machine.

The lot then undergoes X-ray testing. This operation includes six tasks: loading the devices into racks, shooting the X-rays, unloading the devices, developing the X-rays, reading the X-rays for internal defects in the package assembly, and culling the rejects. The lot may experience significant waiting before and after the reading of the X-rays. Three people handle the lot: one who loads and shoots, one who reads, and one who culls.

Electrical testing, the next step, is an important, machine-intensive operation. During an electrical test, the lot is processed by a handler machine that positions each device over a contact with the tester, a computer that performs a number of tests by subjecting the device to a number of different electrical inputs. After the test of that semiconductor is done, the device is fed to an output bin. The test floor contains a number of different types of testers and different handlers. Since each product requires a different combination of handler and tester, significant setup time can be incurred if the operator must find and install a handler to process the lot.

After the initial test, the lot must be burned-in. The devices in the lot are loaded onto boards that hold from 10 to 100 devices each. These boards are placed into ovens, where the devices are subjected to thermal stresses and electrical inputs in order to cause infant mortality. The burn-in period lasts a minimum of 24 hours, although no penalty is associated with keeping the devices in the oven for longer than this time. After the boards are removed from the oven, the devices are unloaded from the boards.

After the burn-in period, the lot returns to a tester and undergoes interim test. Interim test consists of testing at room, low, and high temperatures. For the low temperature test, a bottle of liquid nitrogen must be attached to the handler and the handler must chill the testing environment to the desired temperature. In a test at high temperature, the handler warms the test chamber to the desired temperature.

After the last burn-in and interim test, a final test is conducted. The lot then goes to the assembly facility, where fine and gross leak testing of the package seal is performed. During this time, the lot is out of the test facility's area of control.

The lot returns and moves to brand, where each device is stamped with the company logo. The devices are placed into pans and then sit through a three-hour bake in a kiln. After this step, testing is performed to verify that no damage was done to the devices during brand.

After another trip to assembly, the lot undergoes a visual test, where each device is inspected under magnification for obvious defects in the packaging. The lot enters document review next. At this time, all of the paperwork associated with the lot must be reviewed for completeness.

The lot then undergoes customer source inspection, where a representative of the customer personally supervises the testing of a sample of the lot. In the shipping operation, the devices are placed in boxes and the boxes are addressed, ready for delivery to the customer.

2.2 Semiconductor Scheduling

The manufacturing of semiconductors has received much attention form production planning and scheduling researchers. This section reviews a number of papers that address the issue of semiconductor production planning and scheduling directly or indirectly. As in Uzsoy, Lee, and Martin-Vega (1992a, 1993), the papers are classified into the following topics: production planning; shop floor scheduling: dispatching rules and work release, deterministic scheduling, batch processing, control-theoretic approaches, knowledge-based approaches, and simulation; and performance evaluation: queueing models, and simulation. This review is

intended to display the numerous methods that have been used in the research of production and scheduling issues of semiconductor manufacturing.

2.2.1 Production Planning

A number of authors have addressed the large-scale problem of production planning for the semiconductor industry. The most common approach has been a hierarchical decomposition of the problem.

Leachman (1986) describes a corporate-level production planning system. This system divides the manufacturing process into the stages of wafer fab, probe, assembly and test, linked by inventories. Its model may include multiple facilities. The production processes in a plant are treated as a single entity, and problem complexity is reduced by the aggregation of products that are in the same process at each stage.

Information from corporate databases is used in a linear program with dynamic production functions that capture the relationships between the processes. This yields a production plan. Through a set of linear programs, the aggregate plan is then decomposed into a capacity-feasible weekly start schedule for various facilities. In a later work, this system was implemented by Harris Semiconductor.

Hadavi and Voigt (1987) describe the planning system for a Siemens development wafer wafer fab. In their approach, they create different levels of abstraction for different levels of planning and localize rescheduling while making minimal resource constraints.

The system architecture is a hierarchy of constraint sets that represent different time windows. Quarterly requirements are divided into months, weeks, and days. Arriving orders start feasibility analysis (can it be done?) and then a scheduling heuristic (when will it be done?) that tries to satisfies the constraint sets (based on the hierarchy). Scheduled orders are sequenced by an algorithm that maximizes throughput. A rule-based expert system recovers from disturbances by local rescheduling.

Harrison, Holloway, and Patell (1989) discuss the production planning question and present a case study of National Semiconductor Corporation. They list the following programs that can improve customer service: management information systems; logic and algorithms used to generate delivery quotations and to schedule production; and performance evaluations and incentive systems based on measures of delivery performance. They place the emphasis on operational decision-making, from booking orders to scheduling lots on machines.

The authors also consider performance measurement, which should give the upper management information about the different groups, all of whom are acting to improve their performance on these scales. These measures should encourage mangers to act optimally for the company.

The authors make the following comments about scheduling. Marketing wants commitments to be inviolate; production planners thus want control over factory loading. Managers view scheduling as a MIS problem that provides reliable routine execution. The company must get the right information to the right people and then coerce them to do the right thing.

Golovin (1986) describes various attempts to solve the problem of production planning and factory scheduling. Mathematically, an integer programming problem of the factory scheduling problem considers all costs. The solution philosophies are diverse and involve certain tradeoffs. Mathematical optimization gives the best solution but is computationally costly in data and time and ignores local conditions. Dispatching rules are concerned only with the present. They may make poor scheduling decisions by ignoring global conditions, but they work quickly and cheaply. Just-In-Time maintains a balance of work in the factory and attempts to maintain a high level of quality.

The authors find most promising hierarchical systems that decompose the problem into sets of decisions made at different times: capacity planning, release planning, and lot dispatching. This approach may yield suboptimal policies but it gives control of decisions to the users who are responsible for the results. Scheduling is done under the assumption that sufficient capacity

exists. Dealing with uncertainty in yields and equipment calls for keeping safety stock of standard product and buffers in front of each machine to prevent starvation.

2.2.2 Shop Floor Control

Methods used to control the shop floor vary widely in their approaches to solving the problems encountered there. The basic areas include lot release policies, dispatching rules, deterministic scheduling, control-theoretic approaches, knowledge-based approaches, and simulation.

The production of detailed shop schedules for planning and shop floor control has also been considered by researchers looking for some way to go beyond materials requirements planning, which does not consider capacity constraints, and to search for schedules better than those found by dispatching rules.

Vollmann, Berry, and Whybark (Chapter 5, 1988) discuss three simple ways in which shop floor control can be done: Gantt charts (often created backwards from the job due dates), priority sequencing (dispatching) rules, and finite loading schemes, which create a schedule for the time horizon by simulating the operation of the shop.

Bai and Gershwin (1989) initiate a discussion of all important phenomena in semiconductor fabrication and characterization of all events and scheduling objectives and factory types. They start with the observation that two types of events exist: controllable and uncontrollable. Controllable events are those that the person in charge of scheduling the shop can start. The state of the system is a complete description of the production variables, such as the status of each machine and each worker. The variables in the state are chosen by the scheduler for his purposes, and he may ignore certain inconsequential quantities. Events change the state of the system, and the current state limits the options of the scheduler, who must make a decision based upon the state.

The authors note that schedules in the real world are subject to disruption by uncontrollable events and that schedulers have three weapons to reduce the effects of disruption: real-time

scheduling systems, prediction, and inherently-robust schedules. The first is the most powerful, but the computational effort makes combinatorial optimization impractical; thus, scheduling heuristics are necessary.

The paper includes a summary of the semiconductor manufacturing processes and a detailed description of wafer fabrication. This description includes an explanation of the different machines, workers, and events in the fabrication process. The uncontrollable events are classified into those that are predictable and unpredictable. The latter include machine failure and defective wafers and are the ones that make life difficult for schedulers. The authors also include the constraints on scheduling and the objectives of scheduling, which are the minimization of WIP, throughput, variability in throughput, and costs and the meeting of demand.

Lot release policies and dispatching rules. A number of researchers have realized that the release of work into the shop has large ramifications on the performance of that shop. Dispatching rules are in use in many shops, and some work has been devoted to developing better rules.

Wein (1988) considers the problem of minimizing cycle time in a semiconductor wafer fabrication. His approach is to study the input regulation policy. He studies four alternatives: no control (Poisson arrivals); uniform start policy (constant release rate); constant WIP (closed loop); and workload regulating. This last rule focuses on the heavily-utilized workstations and uses a Brownian network model to approximate a multiclass queueing model with dynamic control capability. A lot is released when the work in the system for a bottleneck machine falls below a certain level. Bottleneck machines use various dispatching rules while other machines use FIFO. The workload regulation rule was found through simulation studies to reduce the mean and variance of cycle time, and the effects of dispatching rules were less significant and varied by system and input type. The study uses data gathered at Hewlett-Packard Technology Research Center in Palo Alto.

Glassey and Resende (1988 a,b) examine a wafer wafer fab with a single bottleneck workstation, single product, and constant demand. Their approach is to control input regulation

by considering global information. The measure of performance is the cycle-time versus throughput curve. Their answer is to release jobs so that they arrive at the bottleneck just in time to avoid starving that machine (starvation avoidance). This requires the determination of the virtual inventory, the amount of work for the bottleneck machine that is there now or will be soon (within the lead time it takes a released job to reach the machine). If this inventory is less than the lead time, release a job. This is similar to a safety-stock inventory policy.

Simulation studies reveal that starvation avoidance results in lower cycle-time vs. throughput curves than other release rules: uniform, fixed-WIP, and workload regulation. Solorzano (1989) describes the implementation at Harris Semiconductor.

Glassey and Petrakian (1989) consider the problem of minimizing the queue of a bottleneck machine in a shop using starvation avoidance input regulation. They do so by using queue predictions and dispatching rules that give higher priority to lots that should encounter a shorter queue on their next visit to the bottleneck. This extra queue prediction computation is intensive, although certain simplifying assumptions are made. Object-oriented simulation studies show good behavior (better than dispatching rules such as SPT and FIFO) in both a one-product wafer fab and a two-product wafer fab.

Wein and Chevalier (1992) take a broader view of the job-shop scheduling problem by considering three dynamic decisions: assigning due-dates to arriving jobs, releasing jobs from backlog to shop floor, sequencing jobs at each workstation. Their objective is to minimize WIP and due-date lead time, subject to an upper bound constraint on fraction tardy. They take a two-step approach: (1) release and sequence jobs to minimize WIP subject to throughput rate; and (2) set due dates that minimize due-date lead time.

The authors propose three principles: (1) while maintaining fraction tardy, average due date lead times can be reduced by dynamically basing due-dates on the status of the backlog and shop floor, the type of arriving job, and the job release and sequencing policies used; (2) without affecting the throughput, WIP can be reduced by regulating the amount of work on the shop floor

for bottleneck stations; and (3) better due-date performance can be achieved by focusing on efficient system performance and ignoring due dates when sequencing.

The proposed job release policy is to inject a customer into the shop whenever the workload at the bottlenecks is at a certain level, determining the customer by a workload balancing input heuristic. Priority sequencing uses dynamic reduced costs from an LP. In step two, due dates are set using rough approximations that follow the spirit of principle 1. Simulation experiments considered a two-machine, two-product shop. The proposed policies beat standard policies.

Lee *et al.* (1993) describe a decision support system for shop-floor control in a test facility. The system uses the Short-Interval Scheduler (SIS) module of COMETS to perform dispatching. The main contribution of the work is the development and implementation of a mechanism that considers sequence-dependent setups while making despatching decisions. This is done by classifying the setups and assigning each operation a setup code representing the setup configuration (determined by handler, temperature, and package type). This allows the operator to select operations with desirable setup characteristics in addition to the due date or operation type allowed by COMETS. This system has been implemented in a test facility and has been running successfully for over two years.

Deterministic scheduling. Under certain conditions, or using simplifying assumptions, controlling the shop floor can be represented by a scheduling problem that can be solved deterministically. The solution to this problem gives a schedule that can be used on the shop floor.

The papers by Uzsoy, Lee, and Martin-Vega (1992b) and Uzsoy *et al.* (1991a, 1991b) consider the scheduling of a semiconductor test facility and the associated single-machine problems. This work addresses back-end and due-date issues. Their first approach is the use of the shifting bottleneck algorithm, which iteratively schedules workcenters based on some measure of criticality.

In another result, they use dynamic programming algorithms and heuristics to minimize maximum lateness and number tardy on a single-machine problem with sequence-dependent setup times. Finally, they minimize maximum lateness on a single tester with a branch and bound algorithm and local search improvements to heuristics. They also describe the prototype implementation of an approximation scheme for an entire test facility (Harris Semiconductor), which may be a practical alternative to dispatching rules.

Graves *et al.* (1983) study the problem of scheduling a production facility with re-entrant product flows for identical products. Their objective is to minimize average cycle time while meeting a target production rate. They develop cyclic schedules that process each operation of a job once every cycle, where the length of the cycle is the reciprocal of the production rate. Thus, in each cycle, one job is started and one job is finished. The scheduling of tasks on machines in the cycle is done with a greedy heuristic that maintains feasibility. The machines in the problem may be multi-channel or batch facilities. The authors compare their rule to the simple FIFO dispatching rule.

Kubiak, Lou, and Wang (1990) consider a re-entrant job shop with a hub machine, the machine to which jobs return repeatedly. They wish to minimize total completion time under the following assumptions: (1) the shortest operation on the bottleneck is longer than any other operation (allowing the single-machine simplification); and (2) the jobs possess a hereditary order, where a smaller total processing time implies a shorter processing time for any operation on hub machine. An optimal well-ordered schedule sequences jobs at the same operation for the hub machine by SPT. They develop a dynamic program and present a heuristic that develops clustered schedules, where groups of jobs scheduled together, finishing one cluster before moving to the next.

Lee, Uzsoy, and Martin-Vega (1992) study burn-in as a single-machine problem. They assume that each lot of devices has been loaded onto a number of boards, and each of these boards forms a job for a burn-in oven. If a batch of jobs is placed into the oven, the entire batch must remain in the oven until all of the jobs have been processed long enough. Thus, the

processing time of the batch equals the maximum processing time of the jobs in the batch. The authors examine the performance measures of maximum tardiness, number of tardy jobs, maximum lateness, and makespan. In this mathematical paper, the authors present dynamic programming algorithms to solve batch scheduling problems with release dates and parallel batch scheduling problems.

Bitran and Tirupati (1988 a,b) consider the problem of epitaxial wafer manufacturing. They identify the bottleneck as the epitaxial growth operation, which takes place in a number of different reactors. Their model is a single-stage parallel unrelated machine problem, with the objectives of makespan and total tardiness. They develop static scheduling heuristics that create schedules in two phases: product group priorities (by workload and due date measures) and then job priorities (by due date) within each group.

They also address the planning problem of assigning reactors to product groups in order to decompose the problem into independent shops. This approach reduces the complexity of the problem and results in the following observation: the choice of heuristic to solve the problem should be guided by the homogeneity of product set and the objective function. An implemented scheduling system provides shop floor schedule for each reactor, job status, lead-time estimates, and reactor load profile. Periodic resolving of model after planning preprocessing creates uniform reactor loads and homogeneous product mixes.

Control-theoretic approaches. In an attempt to find policies that perform better than standard rules, some researchers have created control-theoretic models. The solutions to these models are then used to manage the shop floor.

Bai, Srivatsan, and Gershwin (1990) consider a hierarchical production planning and scheduling system for a semiconductor wafer fabrication facility. They attempt to meet throughput goals while treating random disruptions explicitly. They integrate the scheduler with the system data base. Events in the wafer wafer fab are classified by their frequency and whether they are controllable (starting a lot vs. machine breakdown). Planning hierarchy is organized by these frequencies. Each level has events with the same magnitude of frequency, capacity

constraints, and objectives passed to lower levels. When something with a low frequency happens, the target frequencies of higher-frequency events are recalculated. The real-time scheduling is derived from control theory and treats random events as part of the system. This decomposition allows small but very detailed models. The approach is implemented at the MIT Integrated Circuit Lab. Recent results on this work are covered in Bai and Gershwin (1992).

Bai and Gershwin (1990) cover previous work on scheduling single-part, multiple-part, and re-entrant flow systems using a real-time feedback control algorithm. For single-part systems, the linear programming problem for the feedback control law divides the surplus space into regions where the optimal production rate is constant. There also exists a spot called the hedging point where the surplus is enough to compensate for any disruptions. The feedback controller attempts to drive the system to this point.

The authors measure the system's performance in keeping the amount of surplus close to zero with low buffer inventories. They also attempt to improve behavior by separating the machines from each other to reduce effect of machine failure. Solving a nonlinear program yields an optimal hedging point (minimizes buffer inventory and buffer sizes). The current buffer inventories and the hedging point are then used to calculate the desired production rate for the system. Loading times for machines are calculated heuristically to be close to the optimal production rate.

For multiple-part systems, machines are divided into single-part sub-systems with the same failure and repair rates and capacities proportional to the demand for that part. In re-entrant systems, machines are again divided by part and also by operation, with appropriate capacities calculated.

In Lou and Kager (1989), the authors consider VLSI wafer fabrication with the goal of reducing WIP while following target production and observing machine capacity. Their approach is to use lot release and lot dispatching control rules based on flow rate control, a stochastic optimal control problem. Assuming a continuous flow, the authors divide the shop into workstations and determine a production rate for each according to control rules that consider the

[illegible]

[illegible]

[illegible]

[illegible]

inventories across the floor and predetermined hedging points. The advantages of this model include reducing the dimension (by ignoring machines) and providing dynamic feedback control (by responding to surpluses, down machines). The authors compare their policy to an event-driven simulation using uniform-loading by costs of inventory at all stages, and claim that the flow-rate control reduces costs by 50%.

Gong and Matsuo (1990a) examine a multiperiod production system with random yield, with the objective of minimizing fluctuation of WIP inventory, leading to more predictable system performance. Their approach is to consider control rules for starting product into each stage and raw material. They report that intuitive control rules can be destabilizing. Their new control rule - Minimum Weighted Variance - is the optimal control policy for a stochastic dynamic program with a quadratic objective function that penalizes WIP deviation from targets and infinite time horizon. Rule performs well in systems close to capacity.

Gong and Matsuo (1990b) also consider a problem with multiple products, limited workcenter capacity, rework, and re-entry. Again, they wish to minimize weighted WIP variance, where the weights in the objective are determined by a nonlinear program that minimizes total expected WIP. With sufficient conditions on the variance of yield and rework fractions, the authors find an optimal control policy for the associated dynamic program. An important conclusion is that the development of stable control policies in uncertain environments is challenging and depends upon the yield and rework distributions.

In Ou and Wein (1991), the wafer fabrication system has a single bottleneck, multiple processes with re-entry, and by-products. The authors use for their model a single-server queue with job classes that correspond to different operations. The model's objective is to minimize total cost: the sum of holding costs for WIP (jobs waiting for re-entry) and finished goods and backorder costs. Their approach is to develop a control problem approximation involving Brownian motion. The optimal control policy can be interpreted as the optimal schedule. The authors compare their policy to two state-dependent heuristic policies with a simulation and find that their policy reduces costs.

Knowledge-based approaches. Many researchers have attempted to give managers better control of the shop floor with decision support and expert systems, which incorporate knowledge that can be used to set control policies or schedule machines. Some expert systems can even perform scheduling events themselves.

Adachi, Moodie, and Talavage (1988) consider a production system with re-entrant product flows. They use a simulation model to examine the effect of management decision variables on system performance. The variables are lot size, dispatching rules, start rate, frequency of urgent jobs, and frequency of machine breakdowns. The measures are product mix, throughput, cycle time, and WIP.

The authors report that the start rate was a more important factor than the dispatching rules. The lot sizes also had significant effects. The dispatching rules were important policies only in over-capacity shops. The authors then develop a pattern recognition DSS that, through an iterative procedure, helps the user find a system that matches the user-defined goals. A prototype is implemented on the printed circuit board fabrication line at NEC Corporation.

In Adachi, Moodie, and Talavage (1989), the authors extend their work on production systems with re-entrant product flows. This time, the decision support system includes a rule-based component and simulation model. The simulation model comes from the previous work and evaluates the effect of control variables on measures of inventory, product mix, cycle time, throughput. The results are used to obtain regression coefficients, which are stored in a database for the rule-based component, which organizes the control variables into the hierarchy of start rate, lot size, and priority rule. This DSS was implemented for a printed circuit board fabrication line, resulting in superior control policies to the previous pattern recognition DSS.

Adelsberger and Kanet (1991) report on a new tool in computer-aided manufacturing scheduling: the leitstand, a decision support system with a computer-aided graphical interface. Leitstands include the following components: the graphics component is a electronic Gantt chart; a schedule editor allows a user to easily change an existing production schedule; the data base manager incorporates data from production planning system, engineering, and shop floor and

uses specific knowledge; an evaluation component measures the schedules on objective functions and creates reports; and an advanced automatic schedule generator produces feasible schedules.

Savell, Perez, and Koh (1989) describe scheduling semiconductor wafer production with an expert system that takes advantage of the modularity of workcenters and sophisticated data analysis. The wafer fab is divided into into thirty cells. The expert system, which is an off-the-shelf PC package with some external routines for data manipulation, creates daily schedules. It includes two modules: (1) the priority assignment module uses information from CAM about the operational slack of each lot and knowledge about special lots; and (2) the equipment scheduling module for each cell uses specific knowledge about the cell and the lot priorities to create a schedule for the lots in the cell and those arriving at the cell within a certain time frame. The expert system is implemented in two cells (P-Diffusion and Aluminum Deposition) at Harris Semiconductor.

Sullivan and Fordyce (1990) report on IBM Burlington's Logistics Management System for wafer fabrication. The main function is the shop floor dispatching of lots. It replaces slack lead times with information to handle the coupling of strategic and operational decisions.

The LMS includes real-time lot and machine status and proactive intervention, with an expert system that has knowledge about generating alerts in certain conditions and responding to some of these by making dispatch decisions that consider five conflicting objectives: lot priority, on-time delivery, flow requirements, increasing throughput, meeting engineering specifications. The LMS is implemented in various areas and realizes the importance of accurate data and continual updating to reflect changing environment.

Fordyce *et al.* (1992) discuss the current version of the Logistics Management System (LMS) in place at the IBM Burlington Semiconductor manufacturing site. The emphasis of this paper is on the last of the four tiers of the scheduling decision hierarchy. This is dispatch or short-interval scheduling for periods from one hour to two weeks. This tier contains the decisions concerning the actual manufacturing flow, including the scheduling of an operation. The LMS contains a dispatcher/short-interval-scheduler, that creates zones of control around the bottleneck

[illegible]

[illegible]

> [illegible]

[illegible]

[illegible]

> [illegible]

[illegible]

points, which may a sequence of operations that lots must visit repeatedly. Kanbans within these zones monitor the WIP level for the different passes through the zone. The dispatching is done by something called a Judge. The Judge receives information from goal advocates in order to make its decision, and the different advocates have different goals, including meeting due dates, meeting the daily plan, maintaining low WIP (this is the goal of the kanbans), and increasing machine utilization (this includes minimizing setups).

Hadavi *et al.* (1991) present a distributed architecture for real-time scheduling and describe its implementation in a wafer fabrication factory. The system (ReDS) works to meet management objectives of meeting due dates, reducing WIP and finished goods inventory, reducing cycle times, and maximizing machine utilization. The system abstracts constraints and time into a tree with nodes that correspond to an interval of time; it also abstracts an order into an "essence function" that describes its critical resources. The release policy uses "continuity indices" to reduce cycle times. The modules in the system include a preprocessor to abstract the constraints and orders and a feasibility analysis to release orders. Also in the system are a detailed scheduling module that uses least commitment planning in determining a daily or shift schedule. A sequencer then dispatches the operations scheduled for a time period by using a dynamic sequencing rule that responds in real time to a changing floor.

Rao and Lingaraj (1988) review a number of expert systems for production and operations management decision making. They classify the systems along two dimensions: strategic decisions versus tactical decisions and operations orientation versus resources orientation. Scheduling systems are classified as tactical, operations-oriented applications. They review a number of systems in scheduling, capacity planning, facility layout, process & product design, quality control, aggregate planning, inventory control, and maintenance & reliability. They conclude that expert systems should combine technological knowledge and logistical data in order to be effective.

ISIS (Fox and Smith, 1984) is an expert system for scheduling that uses a hierarchical approach to scheduling: job selection, capacity analysis, resource analysis, and reservation selection. It uses decisions made at each level as constraints for the lower levels.

Opportunistic scheduling is a knowledge-based approach introduced in the OPIS system (Smith, Fox, and Ow, 1986, and Ow and Smith, 1988). In this system, the authors extend the job-oriented scheduler of ISIS with a machine-oriented scheduling procedure. After identifying an initial bottleneck, the search for a good schedule proceeds by directing activity towards the bottleneck subproblems of the job shop scheduling problem. The characteristics of this system include alternative problem decompositions, multiple scheduling heuristics, and multiple problem abstraction.

Sadeh (1991) discusses a system called MICRO-BOSS, another opportunistic scheduler that identifies an initial bottleneck and revises its strategy as new bottlenecks emerge during the construction of the schedule. This system is a more flexible procedure, however, for it can revise its scheduling strategy after each operation. Thus, it avoids having to scheduling large subproblems for a machine or a job.

Bensana, Bel, and Dubois (1988) describe a system called OPAL, a job shop scheduling software that combines three types of knowledge: theoretical knowledge about scheduling problems, empirical knowledge about scheduling heuristics, and practical knowledge of the scheduling environment. They claim that combining artificial intelligence techniques with operations research should be an effective approach to scheduling problems.

OPT and other approaches. OPT (Optimized Production Technology) is a proprietary scheduling system that focuses on the scheduling of the bottleneck resource. The system has been reviewed by a number of authors, including Jacobs (1984), Meleton (1986), Lundrigan (1986), and Vollman (1986). OPT uses a forward finite-loading scheduling procedure to schedule the identified bottleneck. The remaining, non-critical operations are scheduled using an backward infinite-loading procedure. According to Morton (1992), the advantages of OPT are its good solutions to large scheduling problems and its focus on the bottlenecks. Disadvantages

include the inability to do reactive scheduling without resolving the whole problem. Details about OPT can be found in the above articles and in Vollmann, Berry and Whybark (Chapter 20, 1988).

Faaland and Schmitt (1993) use a cost-based system to enhance the planning function of MRP in an assembly shop. They developed a system that can create a detailed schedule of jobs, workers, and work centers for a maker of aircraft audio, power, and light systems. They include inventory holding costs associated with finishing early, opportunity costs associated with late delivery, and payroll costs, since the model includes the cross-training of workers.

Morton (Chapter 16, 1992) describes an early version of bottleneck dynamics called SCHED-STAR (initially reported in Morton *et al.*, 1988). This system iterated lead times and prices over the bottleneck and considered an objective function that included revenue, tardiness, direct completion costs, and holding costs. A number of release and dispatch heuristics were studied on a variety of shop situations, and the authors conclude that bottleneck dynamics (and iterated pricing and lead times) leads to better schedules.

Simulation scheduling. Primarily, simulation models are used to predict the performance of certain policies. However, once the policies are set, the simulation model can be used to create a feasible schedule for the shop floor.

Atherton (1988) states that simulation can be used on all levels of planning. For short-term scheduling, models may perform short-interval and shop-floor scheduling. This form of scheduling considers factory capacity (dispatching rules don't). A validated simulation model, provided with current information on the system status, can use rules to determine what will happen in the short term, providing a shop floor schedule. If the projected lot completions do not meet production requirements, further simulations may be necessary to find a schedule that is close to the goal.

Leachman and Sohoni attack the problem of semiconductor manufacturing using an automated scheduling system and teamwork-fostering management. For every shift, a target is set that considers the real floor. The entire staff meets to identify problems and propose

solutions. The targets are generated by the scheduling system: a simulation model of the wafer fab in BLOCS that is linked to the WIP tracking system, providing it with real-time data. The simulation considers availability but assumes perfect execution. Thus, it is fair and accurate. To make the schedule, it uses logic accepted by the staff, including least slack, FIFO, and starvation avoidance rules and furnace waiting cost analysis.

The authors discuss how a shift proceeds: meetings before and after the shift with explicit incentives for meeting the target. They also review the organizational effectiveness of their approach in terms of motivation, satisfaction, communication and coordination, problem solving capability, acceptance of change.

Najmi and Lozinski (1989) discuss the implementation of the system in two wafer fabs at NCR, Inc. The simulation model is written in BLOCS, an object-oriented language where each physical object or collection of data is a object (composed of data and procedures), and the objects communicate to each other with messages that represent the interaction of objects.

2.2.3 Performance Evaluation

The complexity of semiconductor fabrication facilities makes direct analytical evaluation of them difficult at best. In many cases, researchers use queueing models or simulation models in order to gain some insight on how the system performs in the current or some proposed environment.

Burman *et al.* (1986) discuss the ways that OR tools and techniques are used to analyze IC manufacturing lines: simulation, queueing analysis, and deterministic capacity models. The authors describe a simulation study for direct step on wafer printers in photolithography. The study determined the effect of lot size, number of products, and setup time on capacity and WIP. It also used large-scale simulations to obtain arrival distributions to this area. A deterministic capacity model of a clean room uses the mean number of steps and the mean process time. The analysis took much less computing time than a simulation. The model was useful for estimating a room's capacity and determining the minimal number of machines required for a proposed

program. They conclude that simulation models have the greatest flexibility and best success but take considerable time to develop and run. The other techniques are quicker and, if not quite accurate, do provide estimates that may be verified in simulation.

Queueing models. Queueing models attempt to model the shop floor as a set of servers with service time distributions and possible stochastic routings. The system yields a set of equations that describe its behavior. Because the success of solving these equations is usually undermined by the complexity of the shop being modelled, researchers are forced to make simplifying assumptions.

Chen *et al.* (1988) develop a simple queueing network model to predict performance measures in an research and development wafer fab. The Hewlett-Packard Technology Research Center Silicon Wafer fab serves as the model. The model is a classic job shop, and the performance measures are throughput and cycle time. Machine breakdown rates added to actual service time to create an effective service time. The naive queueing network model includes different types of customers that have different routing distributions. Straightforward formulas from previous researchers yield performance measures. The authors report that the model yielded performance measures that were within 10% of the actual numbers.

Wein (1991) investigates a wafer wafer fab with time-dependent stochastic defects, using a simple queueing theoretic model to determine the relationship of yield and cycle time to throughput. The model is a single-server queueing system with exponential arrival and service times. The author derives a closed form relationship between the mean cycle time and the throughput of good die. In standard models, as the start rate is increased, the throughput increases and asymptotically approaches some upper bound as cycle times increase to infinity. In this model, the cycle times increase without bound but throughput reaches some maximum and then decreases as the increased cycle time leads to a higher defect rate (and thus less yield).

Simulation. Simulation models remain as a popular approach to measuring the performance of manufacturing system, and many packages have been developed. The state of the

art is probably the BLOCS package developed at California-Berkeley and partially described in Najmi and Lozinski (1989). The following papers report on some uses of simulation.

In two papers, Dayhoff and Atherton (1986 a,b) develop simulation models of wafer fabs. They model wafer fab as a queueing network with components for each lot and process flow. Models are used to analyze system performance, measured by its signature: the plots of cycle time, inventory level, and throughput versus start rate. The signature provides a way to gain information from the large quantity of data provided by a simulation by displaying the relationships among different variables. As a specific example, they model a bipolar wafer fab.

They compare the signatures for a number of dispatch schemes for a bipolar wafer fab where products visit the masking station seven times. In the wafer fab, there exist three levels of dispatching: (1) among lots of one product waiting for one process, (2) among process steps of the same product, (3) among products. This paper works at level two; the first level is done by FIFO, third by product priority. The study compares the following dispatch schemes for masking: earlier steps first (a form of longest remaining processing time), later steps first (a form of shortest remaining processing time), round-robin (changing priority). Qualitative analysis shows that earlier steps first failed at higher start rates by building inventory and that the later steps first was better by getting higher throughput with no increase in inventory.

Dayhoff and Atherton (1987) provide definitions of the elements of wafer fabrication and their interactions. They identify the important parameters that govern wafer wafer fab and result from wafer fab operations. The elements defined include work station, process flow, products, wafer lot, process step, batch, batching down, service, dispatch system, rework, and wafer fab graph. They discuss the following types of simulation results: average wait times, queue lengths, inventory, cycle time, equipment utilization, yield, throughput.

Atherton and Pool (1989) discuss the ACHILLES simulation model in a wafer wafer fab of Silicon Systems, Inc. The describe the model and its initialization, calibration, and validation by comparison to actual measures of factory performance. The discuss the use of the model to predict the effect of reducing inventory levels on cycle-times.

Spence and Welter (1987) discuss a project by Stanford University and Motorola Inc. that attempts to improve the performance of a semiconductor wafer fabrication line. The goal is to increase the throughput of the line without sacrificing cycle time. The analysis was done with a Monte Carlo simulation to determine capacity, defined as throughput rate, cycle time, and WIP. The performance was measured on the steady-state cycle-time vs throughput trade-off curve, instead of using transient signatures (c.f. Dayhoff and Atherton). The photolithography cell was studied for its analytical difficulty, the build up of WIP in this cell, and the proposal of system changes. The authors report that adding resources (operators, aligner equipment) reduced cycle time at higher throughput. Reducing reworks, setup times, and the time to wait for repair reduced cycle times at all levels. Larger lot sizes were preferable at higher throughputs.

2.2.4 Summary

From this review, it can be seen that many different approaches have been applied to the problems of production planning and scheduling. However, most of this work addresses the questions raised in wafer fabrication. The primary exceptions are Uzsoy, Lee, and Martin-Vega (1992b), Uzsoy *et al.* (1991a, 1991b), Lee, Uzsoy, and Martin-Vega (1992), and Lee *et al.* (1993) which do explicitly consider the problems of semiconductor test. However, this work does not make use of the new, sophisticated heuristic searches and is not concerned with the ideas of class scheduling and look-ahead and look-behind scheduling.

Although Savell, Perez, and Koh (1989) do develop a system that is look-behind, it is implemented in an off-line system in a wafer fabrication cell. The research in this dissertation, however, investigates the application of both look-behind and look-ahead rules into the current real-time dispatching system of a semiconductor test area.

2.3 Job Shop Scheduling

Before beginning the analysis of scheduling problems, it is useful to review the notation to be used and the basics of scheduling problems. Because even the simplest job shop scheduling problems are NP-complete, the literature consists of different heuristics.

Job shop scheduling, as one of the most difficult scheduling problems, has attracted a lot of attention from researchers. Techniques such as the shifting bottleneck algorithm (Adams, Balas, and Zawack, 1988) or bottleneck dynamics (described in Morton, 1992) concentrate on solving the problem at one machine at a time. Other researchers have studied how well different dispatching rules perform in minimizing makespan and other objective functions. Panwalkar and Iskander (1977) present a list of over 100 rules. Recent studies include Fry, Philipoom, and Blackstone (1988) and Vepsalainen and Morton (1988). Various scheduling systems for shop floor control are reviewed in Section 2.2. More sophisticated look-ahead and look-behind rules have also been introduced; see Section 2.5 for a discussion of these ideas. In this section we review scheduling notation, the shifting bottleneck algorithm, and dispatching rules.

2.3.1 Scheduling Notation

In the general job shop scheduling problem, there exists a set of jobs J_j, $j = 1, ..., n$, and set of machine M_i, $i = 1, ..., m$. Each job J_j has n_j operations (or tasks) O_{ij}, $i = 1, ..., n_j$, where $O_{ij} = k$ if the ith operation of J_j is to be processed on machine M_k. In a flow shop, all $n_j = m$ and $O_{ij} = i$ for all i. In general, each operation has a processing time $p_{ij} \geq 0$. A job can have release dates r_j and due dates d_j and deadlines D_j.

A feasible schedule σ is a plan that determines when each operation is processed on each machine subject to the following constraints: the operations of each job must be performed in order, and no machine can process more than one operation at at time.

For a schedule, certain performance measures can be associated with a job J_j: the completion time C_j, the lateness $L_j = C_j - d_j$, the tardiness $T_j = \max\{L_j, 0\}$, and whether or not

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

the job is tardy, $U_j = 1$ if $T_j > 0$ and 0 otherwise. Objective functions that measure the performance of the schedule include the makespan, $C_{max} = \max \{C_j\}$, the total flowtime $\sum C_j$, and the number of tardy jobs $\sum U_j$.

The scheduling problem is to find a feasible schedule that minimizes the objective function. For the regular performance measures mentioned above, the set of schedules that need to be considered is the set of active schedules, where no operation can be started earlier without causing another operation to be delayed. In this set, there exists a one-to-one correspondence between a schedule and its representation by a sequence of operations for each machine. Thus, a sequence that orders the jobs on each machine can be considered a solution to the problem.

Standard scheduling problems can be classified in a concise way by using the following three-element description: $x / y / z$. The field x describes the machine environment as one-machine, parallel-machine, or shop. The second field (y) describes any constraints or special characteristics of the problem. The third field (z) describes the objective function. Consider the following examples:

$1 / r_j / \sum U_j$ is a one-machine problem where the jobs have release dates and the objective is to minimize the maximum lateness.

$1 / D_j / \sum C_j$ is the one-machine problem where the jobs have deadlines and the objective is to minimize the total flowtime.

$F2 / / C_{max}$ is the two-machine flowshop problem of minimizing the makespan.

$J / / C_{max}$ is the general job shop scheduling problem of minimizing makespan.

2.3.2 Shifting Bottleneck

Adams, Balas, and Zawack (1988) introduce the Shifting Bottleneck procedure to minimize the makespan of job shop scheduling. This algorithm sequences the machines in a job shop successively by identifying the machine that is a bottleneck among the machines not yet sequenced. After the scheduling of the new bottleneck, all of the previously-considered machines

have their schedules readjusted. The individual steps involve finding the solution to one-machine problems.

The authors lean heavily on the disjunctive graph associated with the job shop scheduling problem. In this graph, there exists a node for each operation to be performed as well as dummy start and termination nodes. For each job there exist directed arcs from each operation to its successor and an arc from the start to the first operation and an arc from the last operation to the termination. For each machine, there exist a pair of disjunctive arcs between each pair of operations on this machine. All of the arcs have a length equal to the processing time of the head operation.

If a selection is made for the disjunctive arcs of each machine and the corresponding graph is non-cyclic, a schedule for the entire shop is made by scheduling each job as soon as possible under the constraints imposed by the selections and the job operation sequence. The makespan of the schedule is equal to the longest path from the start node to the termination node.

The Shifting Bottleneck procedure iteratively selects a bottleneck machine, schedules this machine, and readjusts all previously-scheduled machines. Given that a set of machines has been scheduled, procedure determines the bottleneck by considering each unscheduled machine individually. This yields a one-machine problem where the minimization of the makespan for the shop is equivalent to minimizing the maximum lateness of the operations on this machine. The minimization problem is NP-complete, but the authors use the algorithm of Carlier, a branch-and-bound procedure that gives excellent results. The machine that has the largest such maximum lateness is the bottleneck, and that machine is added to the set of scheduled machines.

The local reoptimization procedure considers the set of scheduled machines and performs a number of cycles. Each cycle consists of solving the one machine problem for each machine in turn, using the selections found for all of the other machines.

The authors report that the Shifting Bottleneck procedure performs exceptionally well, including finding the optimal makespan for the notorious 10-job and 10-machine problem of Giffler and Thompson.

2.3.3 Dispatching Rules

Because of the complexity of job shop scheduling, algorithms to find the optimal solution for any arbitrary objective function do not exist. Thus, researchers have studied and schedulers have used dispatching rules to order the jobs waiting for processing at a machine. Most of the research in this area has examined the performance of various dispatching rules, which sequence the jobs that are waiting for a machine according to some statistic that is a function of the jobs' characteristics.

When a machine becomes available, it chooses from among the jobs in its queue by using a rule that sorts the jobs by some criteria. Common dispatching rules employ processing times and due dates in simple rules and complex combinations.

These dispatching rules are sometimes extensions from simple one-machine problems. For instance, the Shortest Processing Time (SPT) algorithm is known to minimize the total flowtime of jobs processed on one machine. The SPT dispatching rule sorts jobs waiting for a machine by the amount of processing time they require on the machine. Also, the Earliest Due Date (EDD) algorithm is known to minimize the maximum lateness of a set of jobs being processed on one machine. The EDD dispatching rule is used in job shop scheduling in an attempt to reduce maximum lateness. While the list of known dispatching rules includes over 100 items, only a handful are commonly used. And most of the rest are combinations of the most common rules.

Day and Hottenstein (1970) present a review of sequencing research, in which they discuss Jackson's Decomposition Principle (1967), which assumes that the arrival times for each job arriving from outside the system are exponentially distributed, the processing times at each machine are exponentially distributed, the jobs are routed to a machine by a fixed probability transition matrix, and the priority rule at each machine is First-Come-First-Served (FCFS).

They also discuss several due date assignment schemes presented by Conway (1965). These are CON (constant from the order to the due date), RAN (random: due date chosen by customer and accepted by salesman), TWK (Total Work Content: the allowable shop time is

proportional to the sum of the processing times of the operations of the job), and NOP (Number of Operations: the allowable shop time is proportional to the number of operations).

The authors also review the previous research on dispatching rules by Conway (1965), who compared SI to 31 other rules and found that it dominated the set of rules tested. It is simpler and easier to implement. However, the biggest objection to SI is the following fact: if the mean time which jobs spend in the system is small, individual jobs (those with long operations) may be intolerably delayed. Thus the variance of the lateness distribution is the basic disadvantage of the SI rule. Conway tried three methods to reduce the variance: (i) Alternating the SI rule with a low variance rule (with respect to flowtime) to periodically clean out the shop, (ii) forcibly truncating the SI rule by imposing a limit on the delay that individual jobs will tolerate; and (iii) dividing jobs into two classes, preferred and regular, as the primary criteria, leaving SI as a secondary criteria.

The authors then move to the COVERT dispatching rule. Buffa (1968) retained the performance of the SI rule and tended to minimize the extreme completion delays of a few orders. Trilling (1966) used a ratio of delay cost rate of a job to the processing time of that job on the machine in question. The job with the highest ratio is dispatched first.

The authors also report on the priority rules employed in industry, where job lateness is a primary concern. Hence, EDD and least slack rules are most used. However, LPT becomes a popular rule because schedulers rank jobs by the index of importance and it is reasonable to expect some positive correlation between importance and processing time.

Holloway and Nelson (1974) study the problem of job shop scheduling with stochastic processing times. Their performance measures are the mean, variance, and maximum of tardiness. For dispatching, the authors use HSP, a multi-pass heuristic that produces delay schedules, HSP-NDT (a non-delay version), DDATE, SPT, SLACK, and a SPT-SLACK combination.

The authors study three problems of different size, tightness, and utilization, where the processing times were constant or from one of three distributions. They report that HSP did well

on low-variance problems and on maximum tardiness. HSP-NDT beat HSP in some situations. Of the non-delay rules, the deterministic problems were a good predictor of relative performance on problems with variability. DDATE improved its relative performance as variability of processing times increased.

Panwalkar and Iskander (1977) list 100 dispatching rules. They categorize these rules into five classes: simple dispatching rules, combinations of simple rules, weighted priority indexes, heuristic scheduling rules, and other rules. The simple dispatching rules are organized into processing time, due date, number of operations, cost, setup time, arrival time, and machine rules. Combination rules sequence jobs by considering first one characteristic and then another. Weighted priority indexes sequence jobs by combining values from different job characteristics (by adding or dividing).

Blackstone, Phillips, and Hogg (1982) state that the best measure of performance is cost effectiveness. Their work covers this objective function and others: tardiness, lateness, flowtime, inventory. They note that analytic measures depend upon Jackson's decomposition, which assumes a FIFO dispatching rule. Other rules lead to interrelated queues and thus researchers use simulation.

For the single-server model, it is known that SI (shortest imminent processing time) minimizes mean flowtime and mean lateness and EDD (earliest due date) minimizes maximum lateness and maximum tardiness. Mean tardiness cannot be optimized by any dispatching rule; thus, the authors conclude that "no single dispatching rule yet developed will optimize delay costs in the job shop environment."

The authors consider a number of dispatching rules. Their first is SI. They report that SI is not affected much by imperfect data, it performed best on mean flowtime, and it minimized the number of tardy jobs and mean lateness for exogenous (constant and random) due date assignment. It also performed better when internal due dates are less than seven times processing time and utilization is high. Modifications to SI to clear jobs that have been waiting include

alternating rules and truncation. Truncation versions seem useful for shops having control over due dates and concerned about very late jobs.

The authors also consider a number of due date rules: EDD, Slack, and Slack-per-operation. Slack-per-operation is the best and, compared to SI or FIFO, produces a smaller variance of job lateness independent of due date assignment. Compared to EDD and slack, it performs best on lateness variance, cost-per-order, job tardiness, number of tardy jobs. Slack may be defined in static or dynamic terms, although rules that use the latter are better. The authors also report that critical ratio rules are also in use.

Among rules that used neither processing time nor due date information are FIFO and NOP (number of operations). The authors report that both perform worse than other good rules. They also conclude that the look ahead-rules NINQ (number in next queue) and WINQ (work in next queue) are not as good as SI in flowtime and inventory criteria and that a value-based rule usually becomes longest processing time, which generally behaves poorly.

The authors report that weighted combinatorial rules are not better than any single rule, although COVERT (delay cost over time remaining) may be useful in shops willing to estimate delay costs. The authors also discuss dispatching heuristics. The look-ahead heuristic LAH allows insertion of idle time in order to process a critical job. Heuristics improve the performance of dispatching rules but their implementation may not be cost effective; a complete study has not been done.

Green and Appel (1981) examine the problem of job shop scheduling by asking the following questions: What traditional dispatching rules do experienced schedulers select? Would dispatch rule selection be influenced by urgency? Would schedulers select a dispatch order based on organizational influence and/or peer pressure? The authors asked schedulers in a number of plants to denote which of the following rules they used: Due Date, Slack, Operations Due Date, Slack per Operations, SPT, FCFS, COVERT, Program in Greatest Trouble (PGT), or Friend Needs a Favor (FNF). The authors report that influence systems affect scheduling. PGT (a

coalition rule) was highly valued, but FNF (an individual rule) was rejected. Traditional and theoretical rules were not highly valued.

Kanet and Hayya (1982) also consider the problem of job shop scheduling. Their paper tries to determine if operation due dates are better. Their comparison is done by running controlled simulation experiments using the following dispatching rules: Earliest Due Date: DDATE and OPNDD; Smallest Job Slack: SLACK and OPSLK; Critical ratio: CR and OPCR. The slack is static; the critical ratio is dynamic: time until due-date over remaining allowance or operation allowance. For critical ratio, each operation must be assigned an allowance. This allowance is proportional to the processing time. The allowance multiple controls the difficulty of meeting due dates.

The authors consider the performance measures of lateness (mean and standard deviation), fraction tardy, conditional mean tardiness, flowtime (mean and standard deviation), and maximum tardiness. The authors report that all rules were outperformed by their operation counterparts on all measures. Introducing operation due dates shifts distribution of job lateness to the left (less) and compresses it. One unexpected result was that OPNDD (due date) outperformed OPSLK (slack) and always minimized maximum tardiness. The CR rule gives lower lateness variances, but the OPCR rule was even better. The authors note that, as the allowance multiple is increased, due dates become larger and DDATE and SLACK rules act more like SPT, minimizing flowtime. For the critical ratio rules, a larger allowance means longer jobs get lower ratios, causing CR to act like LPT.

Baker and Bertrand (1982) take up the problem of single-machine dynamic scheduling and study different due date assignment methods and dispatching rules. They are concerned with average tardiness. The authors introduce a new dispatching rule: the modified due date rule (MDD), where the modified due date is defined as the maximum of the due date and earliest finish date. This rule is a combination of EDD and SPT that implicitly responds to changes in the amount of slack. The authors report that MDD dominates both SPT and EDD on average tardiness.

Baker and Kanet (1983) extend the MDD rule to attack the problem of job shop scheduling. They consider the performance measures of mean job tardiness and proportion of tardy jobs. They introduce the modified operation due date rule (MOD), where the modified operation due date is the maximum of operation due date and earliest operation finish time. The authors also consider other dynamic rules: Slack per operation, Critical ratio, and COVERT under different allowances and utilization levels.

The authors report the following results: MOD performs better than MDD and SPT and is sometimes better than operation due date, critical ratio, and slack per operation rules. The authors thus conclude that MOD is an important dispatching mechanism.

Baker (1984) considers the job shop scheduling problem and attempts to clarify some conflicting results between different rules of dispatching and due-date assignments. He points out that two factors are of primary interest: flowtime and due-date performance. He reports that SPT is the best tactic for reducing mean flowtime. However, no single priority rule dominates performance comparisons. He reports that the critical ratio is best for minimizing conditional mean tardiness (that is, the average tardiness of tardy jobs); SPT is best for number of tardy jobs; however, for mean tardiness, the results are mixed.

Fry, Philipoom, and Blackstone (1988) consider the problem of job shop scheduling with 90% utilization. For the due date assignment, they use the total work method with two different allowances. They consider the following performance measures: mean flowtime, mean tardiness, and root mean square tardiness (designed to punish large tardiness).

They study truncated and alternating versions of the SPT dispatching rule. The authors define the critical percentage (%C) as the percentage of jobs that enter a higher priority queue because they have been waiting a long time. They report that the best rule depends upon the this critical value.

Vepsalainen and Morton (1987, 1988) consider the problem of minimizing weighted tardiness in job shop scheduling. They use the following dispatching rules: FCFS, EDD, Slack per remaining processing time (S/RPT), WSPT, weighted COVERT, and Apparent Tardiness

Cost (ATC). A lead time estimation is necessary for last two rules. Lead time estimates are computed as a fixed multiple of processing times, derived from rough waiting-line analysis and job priority indices, or found from iterating simulations.

The authors first studied the flow shop problem, where ATC and COVERT dominated using any lead-time estimate. Using the iteration estimate resulted in better weighted tardiness. ATC did better on fraction of jobs tardy and inventory measures. In the job shop, under different loads and due date tightnesses, the authors report that the ATC rule did better, followed closely by COVERT. The use of the priority estimates yielded a smaller fraction of jobs tardy.

2.3.4 Summary

In addition to introducing some notation, this section covered two important methods of job shop scheduling: the shifting bottleneck procedure and dispatching rules. Both methods have limitations, however. The shifting bottleneck procedure searches for a schedule and continuously improves upon it. Thus, it is only a local search technique. The dispatching rules that have been studied are mainly short-sighted techniques that do not consider other machines. The research in this dissertation extends this work by considering smart-and-lucky searches and investigating look-ahead and look-behind dispatching rules.

2.4 Flow Shop Scheduling

This dissertation considers a three-machine problem where all jobs are processed on a certain machine and then go to one of two second-stage machines. This problem is similar to a flow shop, and it was useful to review the flow shop scheduling problems that have been previously studied.

The flow shop problem is actually a collection of problems that deal with the minimization of some regular objective function for a set of jobs in a flow shop, where each job consists of a number of different operations that must be processed on a set of machines. The primary feature

of a flow shop is that the sequence of operations is the same for each job. That is, each follows the same path through the shop.

All flow shop analysis starts with Johnson (1954) who studied the minimization of makespan for two-machine flow shop problems and for some special three-machine flow shop problems. His algorithm starts jobs with the smallest tasks on machine 1 as soon as possible and jobs with the smallest tasks on machine 2 as late as possible. For the two- and three- machine cases, it can be shown that only permutation schedules need be considered. Permutation schedules are schedules for the machines in which the sequence of jobs is the same for each machine. Johnson showed that for four machines, passing may be necessary for optimality.

Although most analytical research in shop scheduling has dealt with the makespan objective function, this research is interested in the minimization of total flowtime, also known as the mean completion time, the sum of completion times, the mean time in system, or the total time in system. Therefore, this section concentrates on a number of papers on the minimization of total flowtime, a less-commonly studied objective, and then moves on to other objectives, including maximum lateness, and the number of tardy jobs. Also included in this section are reports on some NP-completeness results and problems with separated setup times.

2.4.1 Makespan

Special cases of the flow shop makespan problem have been studied by a number of researchers, including Mitten (1958), Conway, Maxwell, and Miller (1967), Burns and Rooker (1975), and Szwarc (1977). Garey, Johnson, and Sethi (1976) proved that the general three-machine problem was NP-complete. Problems with release dates, preemption, precedence constraints, or more than three machines have also been studied.

2.4.2 Total Flowtime

Ignall and Schrage (1965) describe a branch-and-bound algorithm for $F2 // \Sigma C_j$ and for $F3 // C_{max}$. For the total flowtime problem, the authors consider two values: the sum of

completion times if the first machine is the only capacity constraint and the sum if the second machine is the capacity constraint. The larger of these two values is the bound at each node.

Krone and Steiglitz (1974) consider the static flow shop scheduling problem with the objective of minimizing the mean flowtime of *n* jobs that must visit *m* machines. They present a heuristic method. The authors note that two previous researchers performed separate sampling experiments of the flow shop scheduling problem. Heller concluded that permutation schedules should be investigated while Nugent found that good schedules tended to be permutation schedules that had allowed some jobs to pass others (a local reversal).

For the flow shop problem, the authors consider semiactive schedules (where no operation can be shifted forward in time) and define a schedule as an array *S* that gives a sequence for each machine. S_{ij} is the job that is *j*th on the *i*th machine. The authors define a two-phase heuristic. In the first phase, the search considers only permutation schedules. In the second phase, the search takes an initial permutation schedule and allows deviations from the uniform ordering. Each phase is a local search, although the neighborhood structures are different for each phase.

Kohler and Steiglitz (1975) study the two-machine flow shop problem with the objective of minimizing the mean flowtime. They present algorithms that they combine with lower bounds that guarantee the accuracy of the heuristics. The authors use the lower bound of Ignall and Schrage. They compute an initial solution by moving down a branch-and-bound tree without backtracking, selecting the node with the lowest lower bound at each stage. They consider different local searches, with either a random start or the above initialization, hillclimbing (first improvement), and one of four neighborhood structures: backward insertion, forward insertion, a double adjacent pair interchange (switches two pairs), and finally the union of the first and third.

The authors report that the good initialization helped the local search perform better than the random starting search.

The authors then present three different branch-and-bound algorithms that differ in the way the nodes are eliminated. Since some of the algorithms exceed computational limits, they determine a bracket that is the relative difference between the final upper bound and the greatest

lower bound. If the goal is to find a solution that is within a certain amount of the optimal, this bracketing idea allows great speedup since a good (sometimes optimal) initial upper bound was used and the lower bound is very close. For a given $r < 1$, the algorithm stops when the lower bound L is greater than or equal to r times the best upper bound U, i.e., $rU < L <$ optimal $< U$.

Miyazaki, Nishiyama, and Hashimoto (1978) present a heuristic algorithm for $F \, / \, / \sum C_j$. Given a sequence, the authors interchange two adjacent jobs and determine the change in flowtime. They then derive a number of conditions that are sufficient to say that the second job should never directly precede the first. Using these different conditions they create temporary sequences and assign each job an ordinal number corresponding to its place in these sequences. The jobs are then sorted by the sum of these values. (more here)

Miyazaki and Nishiyama (1980) extend this work by creating permutation schedules for the weighted mean flowtime flow shop problem. The authors derive some precedence conditions and create an algorithm to generate good solutions. The first analysis considers the effect of interchanging two jobs in the permutation sequence to create. If this switch causes no decrease in the weighted flowtime, then the second job cannot directly precede the first in an optimal sequence. The authors then embark on finding a number of relations for two jobs that are independent of all other jobs and that will determine if a job should precede another.

For their algorithm, the authors evaluate these various relations for each job. If each of the relations yields the same sequence, this sequence is optimal. If not, each job is given a value for each relation that is its performance on the relation (1=best, . . .). These values are added over the relations and the jobs are ordered by these sums.

Szwarc (1983) discusses the mean flowtime flow shop problem and a number of special cases. The author develops two properties that create a precedence relation between two jobs, the first for a fixed presequence, and the second for an arbitrary presequences. Using these properties, the author presents a special case of a 2-job problem, and a case where the final completion times depend only on the completion times on some earlier machine. This last case

holds for the two-machine case where the $t_{1r} \geq t_{2r}$ for all jobs J_r and the case where the columns and rows form a perfect order under dominance.

Ahmadi *et al.* (1989) study the two-machine flow shop subproblem with one or two batch processors as a segment of a larger job shop. They study the objective functions of makespan and total flowtime. They consider two types of processors. The first is a batch processor that has a fixed capacity and the time necessary to process the jobs in a batch is a constant. The other type of processor is called a discrete processor, which refers to a standard single-machine processor. The authors consider the six types two-machine problems that can be formed.

To minimize the makespan of a batch-discrete shop, full batches are optimal, permutation schedules dominate non-permutation schedules, and the optimal solution is to sequence jobs by LPT and fill each batch. For the discrete-batch shop, the analysis is similar, but in this case the jobs should be ordered by SPT. For two-batch-machine-shop, all of the jobs are equivalent. The optimal policy is to completely fill the batches on machine one and to use a dynamic program to fill the batches on machine 2, using the completion times on machine 1 as release dates.

To minimize the total flowtime of a discrete-batch shop, SPT should sequence the jobs on the discrete machine 1. A dynamic program can be used to perform the batch dispatching on machine 2. For the two-batch-machine-shop, the batches on machine 1 should be completely filled and the completion times from this can be used as release dates to batch machine 2, which is dispatched using the same dynamic program. For the problem where a discrete processor is fed by a batch machine, the batches should be completely filled, but the authors prove that this case is strongly NP-complete. They do consider special cases relating the processing times on one machine to another and derive two optimal algorithms.

For the NP-complete problem, the authors propose a heuristic determines the number of shortest batches that are shorter than the batch processing time. To these batches the shortest jobs are dealt. The remaining jobs are sequenced by SPT to form batches. The authors show that the error tends to 1/2 as the number of batches tends to infinity.

Ahmadi and Bagchi (1990) present a lower bound on the total flowtime. Given a partial schedule J_r of r jobs and fixing a machine j, the authors bound the final completion times based on the completion times on machine j. Then, they search over sequences on machine j, considering the sequence constraints and the machine j release dates for each job. This is the NP-hard problem of solving $1/r_j/\Sigma C_j$. The authors solve the problem by allowing preemption. The lower bound is the greatest found by performing this procedure over each of the machines.

Van de Velde (1990) studies the two-machine flow shop problem and minimizes the sum of completion times by using a Lagrangean relaxation. The author decides to relax the precedence constraints between the two operations of a job (instead of the capacity constraints on each machine). Let the vector of multipliers be $\lambda = (\lambda_1, \lambda_2, ..., \lambda_n)$. The relaxed problem is one of minimizing the weighted sum of operation completion times, solved by SWPT on each machine separately. The author, however, wants to restrict the solution to permutation schedules.

Thus, the author reformulates the problem into a linear ordering problem, which is polynomially solvable in certain cases. The author lets all $\lambda_i = c$, where $0 \leq c \leq 1$, and if $c = 0$ or $c = 1$, the problem yields the bounds of Ignall and Schrage. Thus, for c, the relaxation can be solved optimally with a permutation schedule. The author claims that the lower bound as a function of c is continuous, concave, piecewise-linear function, and the maximum can be found in polynomial time.

One partial sequence dominates another if the sum of completion times is smaller and the makespan is also smaller. The author develops a number of sufficient conditions for dominance. Also, if $p_{2i} = p_{2j}$ and $p_{1i} \leq p_{1j}$, then the author proves that there exists an optimal sequence where job i precedes job j.

2.4.3 Maximum Lateness (Lmax)

Masuda, Ishii, and Nishida (1983) find a solvable case of F2 / / L_{max} and present an algorithm for the general problem with a worst case bound. The authors first prove that the EDD sequence is an optimal schedule if for i and j : $d_i \leq d_j$ if and only if min $\{a_i, b_j\} \leq$ min $\{a_j, b_i\}$.

This says that the EDD and Johnson sequence are the same. However, this property does not provide any precedence for the general case, but the authors schedule the jobs according to EDD and give an error bound that is tight asymptotically.

Grabowski, Skubalska, and Smutnicki (1983) the flow shop problem of minimizing maximum lateness with release dates. They analyze two lower bounds, one with one bottleneck machine and a non-bottleneck preceding it and the other with two bottleneck machines and a non-bottleneck between them.

2.4.4 Number of Tardy Jobs

Hariri and Potts (1989) consider a permutation flow shop and develop a lower bound and a branch-and-bound procedure to minimize the number of tardy jobs. This problem is NP-hard. For a given partial sequence and a machine j, we can consider the time this machine is available, disregard the processing before machine j and suppose that the machines after machine j have infinite capacity. The additional processing can be subtracted from the due date and The Moore-Hodgson algorithm applied to find a minimum number of late jobs. Applying this procedure to each of the machines yields a lower bound.

The authors derive another bound using the consistent early set, a set of jobs that is feasibly early for each of the machine subproblems. The size of the smallest consistent late set (a set whose complement is a consistent early set) is the lower bound. The authors extend this to the idea of a consistent feasible early sequence for single-machine and *r*-machine problems.

In generating problems to test their bounds, the authors set due dates that were uniformly distributed in a range proportional to an estimate of the total processing time. The fractions ranged from 0.2 to 1.0. The branch-and-bound algorithm with the simple bound was good enough for the small problems, but the use of consistent sequences yielded the most efficient procedure on the larger problems. The due date range of 0.4 to 0.6 yielded the hardest problems since the range was not great enough to guide sequencing and the due dates were tight.

2.4.5 General Topics

Garey, Johnson, and Sethi (1976) prove a number of NP-completeness proofs for scheduling problems. They do allow zero processing times. The authors begin by discussing some of the terms of NP-theory. All of the proofs will be by transformation from 3-partition, which is NP-complete in the strong sense.

The first proof is that of the F3 / / C_{max} problem, where dummy jobs are created to give slots on the second machine that jobs corresponding the elements of A must fit. The next proof is for the F2 / / $\sum C_j$ problem. Dummy jobs create slots on the second machine that dummy jobs of intermediate length and the jobs corresponding to the partition problem must fill. Long tasks are added to the end to insure that the spacers are completed as soon as possible. Lastly, they prove that J2 / / C_{max} is NP-complete, although this depends upon a job that is re-entrant n times to machines 1 and 2.

Gonzalez and Sahni (1978) consider a number of flow shop and job shop problems, with and without preemption, minimizing makespan and flowtime. The authors extend the makespan results to the preemptive problems.

F3/pmtn/C_{max} is NP-complete even if no job has more than 2 operations. They have jobs with 1-2, 1-3, and 2-3 flows. They transform Partition into this, and the result and proof are the same for non-preemptive jobs. F3/pmtn/C_{max} is strongly NP-complete, including jobs with operations on all three machines.

J2/pmtn/C_{max} is NP-complete if all but two jobs have only two operations. The two other jobs go 1-2-1 and 2-1-2. The result also holds for non-preemptive jobs. J2/pmtn/C_{max} is strongly NP-complete with one job that visits each of the two machines n times.

The authors move to some approximation algorithms and bounds. For the total flowtime job shop problem, the bound on the ratio of the flowtimes for any busy schedule to the optimal is m. And this holds if the busy schedule is SPT. The ratio for makespan is also bounded by m. A variation of Johnson's algorithm that considers pairs of machines has an error bound of $m / 2$.

Lageweg, Lenstra, and Rinnooy Kan (1978) report on a number of different lower bounds and a classification of them as well as showing that a new lower bound for the makespan problem is better than the rest.

They present two dominance criteria and then move to the lower bounds, which relax the problem by allowing some machines to have infinite capacity. The unit-capacity machines are called bottlenecks and the infinite-capacity one are non-bottlenecks. The authors limit consideration to at most two bottleneck machines M_u and M_v and combine consecutive non-bottlenecks into one by summing the processing times for each job. Additionally, a non-bottleneck can be eliminated by adding the minimum processing requirement on this machine to the lower bound. This leads to nine different schemes (excluding symmetric ones). The authors describe each of the nine and analyze the effort required for each. They describe some upper bound calculations.

They test all of the lower bounds on problems with six jobs and three, five, or eight machines. The better ones were tested on larger problems with 10 to 50 jobs and 3 to 5 machines. The best results were obtained using elimination criteria and the two bottleneck machine bound with a non-bottleneck between them and the head and tail non-bottleneck machines removed.

Some researchers have looked at the flexible flowshop problem, where there may exist two or more parallel machines at a stage of the flow. Wittrock (1988) attempts to minimize the makespan and the queueing time. He creates three subproblems: machine allocation, sequencing, and timing. In order to simultaneously minimize makespan and queueing, he develops a heuristic procedure to greedily sequence the jobs given an allocation of the jobs to machines. He applies the LPT heuristic to do this allocation at each stage. Timing consists of loading the parts into the system as late as possible while not delaying any subsequent operation. He also considers buffer limits.

Sriskandarajah and Sethi (1989) consider a number of simple heuristics for the flexible flowshop problem. They derive worst-case error bounds for list scheduling and a Johnson-like sequence on the two-stage case where the first stage has only one machine. Their best relative

error bound is 2. They also examine the problem where each of the two stages has *m* machines. Their heuristic creates *m* two-machine flowshops and uses list scheduling and LPT to allocate the jobs to the different shops. Each shop is sequenced by Johnson's algorithm.

Gupta (1988) and Gupta and Tunc (1991) study the special two-stage cases where one of the stages has only one machine. Both papers propose heuristics that sequence the jobs (by Johnson's or some algorithm rule) and then assign the jobs to the parallel machines. Lower bounds on the makespan and the results of empirical testing are also discussed.

Lee, Cheng, and Lin (1993) study a three-machine problem where each job consists of two tasks that are assembled in a third operation. Each of the two first-stage tasks is processed on a different machine and the tasks can be processed in parallel. The assembly task is performed on the third machine and cannot begin until both first-stage tasks are finished. They show that the problem is NP-complete. The authors provide some restricted versions that are polynomially complete. These include cases where one of the tasks dominates the other, where the assembly tasks is dominated by both first-stage tasks, and where the assembly tasks dominates the first-stage tasks. The authors present a branch-and-bound approach that uses the cases presented to trim the search tree. The authors present heuristics (with error bounds) that create new first-stage tasks from the job data and scheduling according to Johnson's algorithm to minimize the makespan of a two-machine flowshop. The best relative error bound is 1/3.

This three-machine problem is therefore closely related to problems previously studied, but the special structure of the problem (the different flows) leads to interesting twists on these results.

Lee and Herrmann (1993) look at the three-machine look-behind problem with the objective of minimizing the makespan. They derive results similar to the ones we derive for the three-machine look-ahead case.

2.4.6 Summary

The papers covered in this section provide a number of ideas that may be useful in the three-machine problems to be studied. The three-machine problem does not fit into the flow shop model, and thus work must be done on how to modify the above approaches. Also, the NP-completeness proofs do not apply, and the NP-completeness of the three-machine problem must be proved. The flexible flowshop models and the assembly flowshop model of Lee, Cheng, and Lin (1993) may be provide some results on minimizing makespan that can be applied, although the problem is fundamentally different for the flowtime objective.

2.5 Look-ahead and Look-behind Scheduling

This research includes in its investigation of the complicated job shop scheduling problem the use of more sophisticated dispatching techniques that consider more information in their decision-making. This research tries to be precise in its use of the terms *look-ahead* and *look-behind*. The idea of looking at the other machines in the shop when dispatching is not well-researched, although the work-in-next-queue and number-in-next-queue dispatching rules mentioned by Panwalker and Iskander (1977) are simple look-ahead rules.

Look-behind rules consider the jobs that will be arriving soon (called *x-dispatch* by Morton and Ramnath, 1992). Look-ahead rules consider information about the machines downstream in the flow. This includes the work-in-next-queue and the number-in-next-queue rules of Panwalkar and Iskander (1977), bottleneck starvation avoidance (Glassey and Petrakian, 1989), bottleneck dynamics (described in Morton, 1992), and lot release policies that look-ahead to the bottleneck (Wein, 1988; Glassey and Resende, 1988a; and Leachman, Solorzano, and Glassey, 1988). A three-machine look-behind scheduling problems similar to problems studied in this dissertation is considered in Lee and Herrmann (1993).

Also, Blackstone, Phillips, and Hogg (1982) report on a heuristic that is called look-ahead but would be classified here as look-behind, and Savell, Perez, and Koh (1989) did develop a look-behind system.

More specifically, look-ahead models consider the states of the machines where the jobs will be processed after being processed on the machine being scheduled. Information about the workload of other machines may be useful in balancing the flow of product through the shop. Look-behind models consider the machines the precede the machine being scheduled in the flow and the jobs being processed on these machines. Some of these machines will be processing jobs that will next need processing at the machine being scheduled. If the times that these jobs will complete is known, these times form release dates to the machine being scheduled, and the scheduling decision can explicitly consider these imminent arrivals.

Other researchers have studied procedures that they called look-ahead scheduling, but the problem setting or interpretation is slightly different.

Vepsalainen and Morton (1987) called their weighted COVERT and apparent tardiness cost rules "look-ahead" since they are concerned with the remaining waiting time of a job. They use, however, average waiting times without looking at the current queues.

Koulamas and Smith (1988) are concerned with the scheduling of jobs on machines that are attended by a server that must unload a job from a machine and load another job onto the machine. Interference results from one machine finishing while the server is busy at another machine. This interference degrades the system performance by preventing the machine from being maximally utilized.

The authors study a two-machine system where each machine has a distinct set of job types arriving to it. The authors propose a look-ahead rule for scheduling a machine that considers the state of the other machine when deciding what job to sequence next. The rule attempts to schedule a job whose completion will not interfere with the completion of the job on the other machine.

This work is similar to the definition of look-ahead in this paper, although its objective (to minimize interference) is not related to the completion-time objectives being studied.

Zeestraten (1990) is concerned with minimizing makespan in a job shop with routing flexibility. That is, some operations may be able to choose from more than one machine for their processing. The author defines a look-ahead rule as one that considers the entire state of the system and all of the unscheduled operations and creates a partial schedule that specifies a few operations for each machine. This schedule is followed until another scheduling decision is made. At this point, the procedure is repeated, using the new information about state of the system.

The look-ahead rule is so called because it searches through the states that the system could reach in a period that is approximately twice the average cycle time. That is, it looks ahead in time without attempting to schedule all of the remaining operations. Thus, it falls somewhere between fixed schedules and real-time dispatching.

This type of scheduling is actually more global in nature, since it considers the entire shop when scheduling. It is not looking at specific machines.

In summary, although scattered look-ahead and look-behind techniques have been considered, this research attempts to categorize these ideas, conduct analysis of look-ahead and look-behind scheduling problems in order to derive good rules, and integrate the results into a job shop scheduling environment.

2.6 Class Scheduling

Manufacturing often involves machines that process different product types, and this phenomenon can be modelled as a class scheduling problem, a topic mentioned in the introduction (Section 1.4). A number of researchers have studied the class scheduling case of sequence-dependent setup times. Examples reported by Monma and Potts (1989) include paint production machines that are cleaned between the production of different colors, a computer

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

system that must load the appropriate compiler for a type of task, and a limited labor force with workers switching between two or more machines.

Bruno and Downey (1978) study class scheduling problems with deadlines. Their problem is the single-machine scheduling of classes of tasks with deadlines and a setup time or changeover cost between classes. They prove that, with setup times, the question of finding a feasible schedule is NP-complete. With changeover costs, the problem of finding a minimal cost feasible schedule is also NP-complete.

Monma and Potts (1989) present complexity results for class scheduling problems without deadlines. The assume that the class setups satisfy the triangle property. They show that minimizing makespan, maximum lateness, the number of tardy jobs, and the unweighted and weighted flowtime is NP-complete, although they discuss a number of optimal properties and dynamic programming algorithms that are polynomial in number of jobs but exponential in the number of batches. They also show that the corresponding parallel machine problems are NP-complete. The authors conclude that the design and analysis of heuristics for these problems is important.

Ahn and Hyun (1990) study the problem of minimizing flowtime and present a dynamic programming algorithm that is similar to those of Monma and Potts. This algorithm is exponential in the number of classes, and the authors develop an iterative improvement heuristic that finds near-optimal schedules.

Sahney (1972) considers the problem of scheduling one worker to operate two machines in order to minimize the flowtime of jobs that need processing on one of the two machines. Sahney derives a number of optimal properties and uses these to derive an intuitive branch-and-bound algorithm for the problem.

Gupta (1984) also studies the two-class scheduling problem and derives an $O(n \log n)$ algorithm to minimize flowtime. Potts (1991) presents an example that Gupta's algorithm does not solve and goes on to describe an $O(n^2)$ dynamic program to minimize flowtime and an $O(n^3)$ algorithm to minimize weighted flowtime.

Coffman, Nozari, and Yannakakis (1989) consider a class scheduling problem with two subassembly part types. A product consists of two parts, one of each type, both made on the same machine, and the product cannot be delivered until both parts are finished. Their objective is to minimize the flowtime of delivered products, and using properties of an optimal schedules, they authors create an $O(\sqrt{n})$ algorithm. The authors describe extensions to multiple copies, limits on the number of changeovers, and limits on batch size as solvable cases without details.

Mason and Anderson (1991) study the one-machine class scheduling problem with the objective of minimizing weighted completion times under sequence-independent setups that fulfill the triangle inequality. By using properties of an optimal sequence and aggregating jobs into composite jobs, the authors develop a brand-and-bound algorithm using their dominance criteria and lower bound.

Dobson, Karmarkar, and Rummel (1987) consider a class scheduling problem with the objective of minimizing flowtime under both the item-flow and batch-flow delivery schedules. The batch (or class) setups are sequence-independent and the processing times are equal for all parts (jobs) in one part type (class). They formulate integer programs for both problems, solving the item-flow problem and single-product batch-flow problems optimally. For the multiple-product batch flow, the authors can only provide some heuristics based on their other results.

Dobson, Karmarkar, and Rummel (1989) consider the above single-item problems on uniform parallel machines. They assume that the work is continuously divisible and determine the amount of work to allocate to each machine by solving a convex programming problem. The authors also apply their results to solve the item-flow problem.

Woodruff and Spearman (1992) consider an interesting class scheduling problem where the objective is to maximize the profit of feasible schedules. That is, jobs must be completed by their deadlines. The profit calculation includes the value of jobs that are not required to be processed but add some revenue and two different costs: holding and setup. The authors search for good solutions with a tabu search. Discussion of this search can be found in Section 2.8.3.

A more typical problem is studied by Ho (1992), who examines the problem of minimizing the number of tardy jobs where the jobs fall into exactly two classes. He develops and efficient branch-and-bound algorithm.

Gupta (1988) studies the class scheduling problem of minimizing the total flowtime. He develops a heuristic that is based upon the standard shortest processing time rule. Empirical testing shows that the heuristic produces good results on small problems.

A more sophisticated class scheduling problem is the minimization of the maximum lateness of a set of jobs with non-zero release dates. This problem is considered by Schutten and Zijm (1993), who develop a branch-and-bound algorithm and a tabu search over the sequences of jobs. They report good results on problems with up to 50 jobs.

In summary, this body of research does not yet include much work on problems with additional criteria, such as deadlines or release dates, or on shop problems. Moreover, this research has concentrated on branch-and-bound techniques and optimal algorithms for problems with just two classes. Little work has been done on heuristics for problems with an arbitrary number of job classes or on the use of smart-and-lucky searches to solve class scheduling problems.

2.7 Some One-machine Problems

In order to gain insight into new dispatching rules that may be useful in the job shop scheduling question, some one-machine problems will be investigated. The three one-machine problems reviewed in this section are the problem of minimizing total flowtime with deadline constraints, the problem of minimizing the number of tardy jobs subject to matching release dates, and the problem of minimizing the total flowtime of jobs with release dates. These questions will be the first to be studied as class-scheduling problems.

2.7.1 Constrained Flowtime

The first one machine problem studied in the preliminary work is a class scheduling extension of $1 / D_j / \Sigma C_j$, the constrained flowtime problem first studied by Smith (1956).

Smith, in this paper, proves the optimality of ordering the jobs by Shortest Processing Time to minimize the total flowtime problem $1 / / \Sigma C_j$. He extends this to the problem where the jobs all have deadlines that must be met.

The makespan of the jobs is known, and following the SPT property of minimizing total flowtime, the algorithm tries to schedule the longest job to end at this time. However, the longest job's deadline may be less than the makespan, in which case the job is not available to end at this time. So, the algorithm searches for the longest job that can end at this time and schedules it. The algorithm then moves to the start time of this scheduled job. This time is now the completion time of the remaining unscheduled jobs, and the algorithm again searches for the longest available job. This algorithm is hereafter referred to as Smith's algorithm.

The weighted problem is strongly NP-complete (Lenstra, Rinnooy Kan, and Brucker, 1977), and different elimination criteria and branch-and-bound techniques have appeared. Potts and van Wassenhove (1983) study the Lagrangean relaxation of the deadline constraints, leading to the discovery of optimal solutions. Work by Posner (1985) and Bagchi and Ahmadi (1987) present improved varieties of this bound. Many other problems have been studied; see Herrmann, Lee, and Snowdon (1993) for a survey of dual criteria problems.

2.7.2 Release and Due Dates

The other one-machine problem in the preliminary results is the look-behind problem $1 / r_j / \Sigma U_j$, which is a strongly NP-complete problem in general (Lawler, 1982). The problem has been solved optimally if all release dates are zero by the Moore-Hodgson algorithm (Moore, 1968). The problem has been solved optimally by Kise, Ibaraki, and Mine (1978) if the release

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

and due dates match ($r_j < r_k$ implies $d_j \leq d_k$). They present an O(n^2) algorithm (Kise's algorithm, described below) in this case.

Kise's algorithm orders the jobs by their release and due dates (a non-ambiguous ordering since the dates must match). The algorithm is an extension of Moore-Hodgson algorithm for minimizing the number of late jobs. Each job is scheduled after the partial schedule of on-time jobs while maintaining release availability. If the new job is tardy, the algorithm searches the on-time jobs for the job whose removal leaves the shortest schedule of on-time jobs. The removed job is made tardy and will be processed with the other tardy jobs after the on-time jobs. In this manner, the algorithm finds the largest subset of the jobs that can be delivered on-time. These jobs are scheduled in order of their release and due dates.

Unlike the Moore-Hodgson algorithm, the subalgorithm that removes a job cannot just choose the longest job, since the presence of release dates limits how the removal a single job affects the completion times of later tasks. They present an efficient way to determine which job should be removed.

2.7.3 Flowtime and Release Dates

A last one-machine problem that may yield some good results when studied as a class scheduling problem is closely related to the constrained flowtime problem. This is the problem of minimizing total flowtime subject to job release dates. Written as $1 / r_j / \sum C_j$, this problem is a strongly NP-complete question, as shown by Lenstra, Rinnooy Kan, and Brucker (1977).

Dessouky and Deogun (1981) present a branch-and-bound algorithm with a lower bound and dominance properties used to prune the search tree. They list a number of dominance properties that hold for a given partial sequence of the jobs. Their lower bound is derived from the EFT schedule but starts the selected job at the first release time. They can solve problems with up to 50 jobs. They also report that the EFT sequence generally finds good solutions (within 3% of optimal).

Bianco and Ricciardelli (1982) consider the weighted version of the problem. Let the weighted processing time be the processing time divided by the weight of the job. They present an improved branch-and-bound algorithm with further dominance properties for nodes with a partial schedule. They compute a lower bound by allowing preemption. The hardest 10-job problems to solve were those with a larger range of weights and a maximum release date near the expected sum of processing times (that is, the average interarrival and processing times were nearly equal).

Hariri and Potts (1983) also attack the problem of minimizing weighted flowtime with release dates. They use a Lagrangean relaxation of the constraints $C_j \geq r_j + p_j$ to find a lower bound. They make use of previous dominance properties and add another that measures the effect of interchanging two consecutive jobs. They solve problems with 10 to 50 jobs, and the hardest problems (some of which remain unsolved) were those with a range of release dates approximately the same as the expected sum of processing times.

Gazmuri (1985) studies the question probabilistically. He develops two cases. In the undersaturated case, where the expected processing time is strictly less than the expected time between release dates, the author develops an algorithm that partitions the jobs into smaller sets and schedules each of the sets optimally. The algorithm for the oversaturated case starts with the optimal preemptive schedule and patches preempted jobs by shifting the delayed segments to the left until the job is whole. For both cases, the heuristic is asymptotically optimal as the number of jobs goes to infinity.

Liu and MacCarthy (1991) use both mixed-integer linear programming and branch-and-bound techniques to solve the problem. They present a number of heuristics that are different priority rules and report that the heuristics generally find near-optimal solutions (within 1%) with little computational effort until the problem sizes exceed 100 jobs. The MILP can solve problems with only 10 jobs, the branch-and-bound procedure problems with 25 jobs.

Finally, Rinaldi and Sassano (1977) also report on a branch-and-bound technique for the weighted problem.

2.8 Smart-and-lucky Searches

In this section we will discuss a class of searches that have been receiving much attention from researchers recently. This class include tabu search, simulated annealing, and genetic algorithms. We will use the last of these in the research into different scheduling problems.

2.8.1 Introduction

One approach to difficult scheduling problems such as job shop scheduling is local search, an iterative procedure that moves from solution to solution until it finds a local optimum. Two examples are hillclimbing and steepest descent. The hillclimbing algorithm chooses a nearby point at random and moves there if the point improves the value of the objective function. The steepest descent procedure examines the entire neighborhood of an incumbent point and selects the point (if any exist) that is most improving.

Heuristic searches (or probabilistic search heuristics) attempt to improve upon the primary problem of these simple searches: convergence to local optima. We can use the term *smart-and-lucky* to describe these more complex searches: they are smart enough to escape from most local optima; they still must be lucky, however, in order to find the global optimum. Simulated annealing, the most popular of the methods described in this work, was developed independently by Kirkpatrick, Gelatt, and Vecchi (1983) and Cerny (1985). Glover claims that tabu search, which is also widely used, goes back to a paper of his from 1977. Holland developed the ideas of genetic algorithms, which have had the least success so far, in his 1975 book. This section reviews the basic concepts of these searches and a number of scheduling applications.

2.8.2 Simulated Annealing

Simulated annealing (SA) is a variant of the hill climbing algorithm. As mentioned before, both Kirkpatrick, Gelatt, and Vecchi (1983) and Cerny (1985) produced the initial papers on the

simulated annealing algorithm, so called because the algorithm views optimization as a process analogous to physical annealing, the cooling of a system until it reaches a low-energy state.

In annealing, as a system cools, configurations occur with weight dependent upon their energy and the system's temperature. In 1953 Metropolis *et al.* developed a statistical simulation of an annealing system. This algorithm randomly shifted the system from one configuration to another, using the physical laws that governed behavior to guide the procedure. Kirkpatrick, Gelatt, and Vecchi (and Cerny) modified his procedure to solve optimization problems. The crucial element was the acceptance of a non-improving move with a probability that was a function of the difference D in objective function and some cooling constant T. The standard formula was $e^{-D/T}$. This acceptance of a non-improving move is what differentiates a simulated annealing from hillclimbing.

Kirkpatrick, Gelatt, and Vecchi (1983) list four ingredients for a simulated annealing algorithm: a precise description of system (that is, a solution space), the random generation of moves, a quantitative objective function, and an annealing schedule. Simulated annealing makes a move by picking from a set of possible operations that can be applied to the incumbent solution. Usually, the choice is made randomly, although the set may be ordered somehow. At a given temperature, the algorithm may stop when some equilibrium or maximum number of moves is reached. Then the algorithm reduces the temperature geometrically or linearly. The initialization of the algorithm with a starting point may be random or may use a heuristic.

It can be proved that simulated annealing will converge to a global optimum. Aarts and van Laarhoven (1985) use a homogeneous Markov chain that reaches equilibrium at each temperature to prove this. They use a general form of acceptance probabilities, of which the standard exponential is a special case. However, reaching equilibrium at a temperature may require an exponential number of moves. They avoid the exponential chain length by defining quasiequilibrium conditions and conclude with a polynomial algorithm.

Kirkpatrick, Gelatt, and Vecchi (1983) studies a number of chip design problems and the traveling salesman problem. In his example, he places the cities in nine clumps and examines the

effect of temperature on the solutions found. At high temperature, the algorithm minimizes the number of long jumps; at a medium temperature, the coarse structure changes but stays minimal; and at low temperatures, the procedure improves the solutions at the small scale. Cerny also studies the traveling salesman problem.

Van Laarhoven, Aarts, and Lenstra (1988) study the job shop scheduling problem by making use of the disjunctive graph, where the makespan of a solution is the length of the longest path. The disjunctive graph has an arc for each precedence among operations for a job and an (undirected) edge connecting those operations on different jobs that use the same machine. A schedule can be found by directing the edges to form an acyclic graph. Given a feasible solution, any swap of a critical disjunctive edge will form another feasible solution. This becomes their move, and they claim that their simulated annealing is as good as shifting bottleneck and simpler.

A different type of simulated annealing is included in the Matsuo, Suh, and Sullivan papers (1988, 1989) on controlled search simulated annealing (CSSA). In this approach, the algorithm uses a good initialization, independent acceptance probabilities, a low initial acceptance probability, and a sequential search of smaller neighborhoods. The motivation for acceptance probabilities independent of the change in objective is its use in empirical testing of different cooling schedules. Their first paper addresses the single-machine weighted tardiness problem. The CSSA uses a fixed number of iterations and its move swaps an adjacent pair. The authors compare different cooling schedules and claim that the CSSA is a better procedure than standard SA.

In the paper on job shop scheduling, the authors use the shifting bottleneck algorithm for initialization. The CSSA identifies the critical path and then switches two operations on the critical path with look-back and look-ahead options that swap other operations to improve makespan. If a non-improving move is not accepted, they perform a local search to find a possible improving solution. They claim better makespans with the same effort as shifting bottleneck.

For flow shops, Ogbu and Smith (1990) present a simulated annealing that they call "probabilistic-exhaustive" because it uses independent acceptance probabilities with a random search of the entire neighborhood, moving to the last point accepted. They minimize makespan, and their moves are pair swaps and insertions. They get better processing time than the standard SA and better results than repetitive local searches. Vakharia and Chang (1990) look at a flow shop with jobs in families and family setup times. They also use independent acceptance probabilities. Their moves swap jobs within a family or swap families. They use different initializations and compare their search to other heuristics.

2.8.3 Tabu Search

Tabu search (TS) is a variant of steepest descent. Glover (1989, 1990) presents a good discussion of tabu searches. Given an incumbent solution, a TS searches the neighborhood of this solution, finding the best allowable move. A TS allows bad move away from local optima, prohibits moves which lead backwards through short-term memory (the tabu list) and has an aspiration level to override the tabu list under certain conditions (usually best found). A tabu search works because the tabu list forces the search to explore new areas of the solution space. The short-term aspect of the memory and the aspiration level allow the search to get to a global optimum however.

Table 2.1. Outline of a Simple Tabu Search.

1. Pick initial x. let $x' = x$. T, the tabu list, is empty.
2. Let $S(x)$ be the neighborhood of x. Take s' as best member of $S(x) \setminus T$.
3. Let $x = s'(x)$. If $c(x) < c(x')$, $x' = x$.
4. Check stopping criteria. Update T. Go to 2.

The best member is usually that which gives the best value of the objective function $c(x)$. The stopping criteria may be a total number of moves or a number of moves since the last best.

[illegible]

[illegible]

[illegible]

$$[illegible]$$

[illegible]

Updating the tabu list T is the crucial part of a tabu search. Usually this means adding moves that would reverse the step just taken and removing old tabu moves.

Laguna, Barnes, and Glover (1989) investigate a tabu search for a single-machine scheduling problem with linear delay penalties and setup cost dependencies. They use a heuristic initialization and insertion and swap moves. They claim results much better than a branch-and-bound.

Barnes and Chambers (1991) investigate the use of a tabu search to solve the job shop scheduling problem of minimizing makespan. The authors describe an improved approach that uses an actual makespan change when evaluating new schedules instead of a projected change.

The authors compare their job shop scheduling approach to Applegate and Cook's shuffle algorithm and the Shifting Bottleneck algorithm of Adams, Balas, and Zawack. They observe that their tabu search achieves better makespans in most cases over a range of problem sizes.

Barnes and Laguna (1992) examine the multiple-machine weighted flowtime problem with a similar tabu search. This problem reduces to a partition problem, and their search does prevent non-improving swaps: they claim that swaps are too small to move through the solution space effectively. Again, they claim results that beat a branch and bound approach.

Widmer and Hertz (1989) take up the flow shop and use permutation schedules. They define a distance between two jobs as the approximate increase in makespan and perform a TS upon the open TSP. A move in the solution space is the swap of a pair. They get slightly better makespans than six heuristics at the cost of terrible processing times.

An ambitious paper by Malek *et al.* (1989) examines parallel and serial tabu searches and simulated annealing on the traveling salesman problem. For each search, a move was a 2-opt (subsequence reversal). The parallel runs communicate periodically, sharing good solutions. Parallel SA performed a quick annealing on each processor and achieved superlinear speedup. According to the authors, parallel TS performed best, but the simulated annealing was more robust (less sensitive to parameter changes).

In an interesting article, Woodruff and Spearman (1992) study the problem of maximizing the profit gained from scheduling a set of jobs on one machine. The authors claim that the one-machine problem has significance for shops with a bottleneck operation, where the sequencing of the bottleneck determines how the shop performs. These jobs include jobs that are required in some sense, say to meet specific orders, and jobs that are filler, in that they are not required now, but some profit is realized by producing them. The jobs have due dates, and the schedule should be feasible with respect to these. The costs include the holding costs of finishing jobs too early and setup costs incurred when the machine must be switched from processing jobs of one family to those of another. Thus, this problem is a class scheduling problem, and it is an NP-complete question just to ask if there exists a feasible schedule.

The authors show that if the holding costs are zero, it is optimal to order the jobs in each family by EDD. If the holding costs are positive, this EDD within a family is still used as a heuristic in order to simplify the problem of finding a good schedule.

The authors use a tabu search to search the solution space for good sequences. The moves of the tabu search are insertions of jobs while maintaining EDD within a family. Complications are added by the presence of filler jobs that are not required to be in the schedule. The authors use a diversification parameter for two reasons: first, the parameter diversifies the search by being included in a modified cost function that provides a penalty for infeasible solutions while allowing the search to use these as passes into new areas. Secondly, this parameter allows the search to optimize its performance. This is done by initially performing tabu searches over a number of different values of the diversification parameter and then continuing the search with the best values.

Empirical testing of the algorithm on data motivated by an actual manufacturing environment showed that the search is a useful method of finding good solutions for this problem.

Glover, Taillard, and de Werra (1991) describe the main aspects of tabu search and discuss various refinements that may lead to a new generation of search. These refinements occur on the tactical, technical, and computational levels. Tactical improvements are concerned with

improving the neighborhood structure and defined moves. This may include the use of learning approaches, shifting penalty tactics, and strategic oscillation.

Technical improvements focus on the issues of neighborhood size, tabu list structures, and aspiration conditions. Computational improvements include reducing the time required to compute the objective function over the neighborhood and the parallelization of the search.

Laguna and Glover (1991) use target analysis to promote diversification in a tabu search used to find good solutions to a single machine problem with linear delay and setup costs. The tabu search uses both swap and insert moves.

The goal of using target analysis is to teach the tabu search to use good rules. The authors describe five phases of target analysis. The first phase is to take some specific problems and find very good solutions to those problems. In the second phase, with these very good solutions as targets, the problems are solved again and information is gathered on how well the current decision rules lead to the target. In the third phase, this information is integrated into a master decision rule. The fourth phase finds good parameter values for this master. Finally, the master is applied to the original problems to verify its merit.

Reeves (1993) examines how the definition of the neighborhood in a tabu search affects the balance between exploration and exploitation (or diversification and intensification). Given the proper balance, he finds that tabu search is a more efficient procedure than simulated annealing for the permutation flowshop problem.

2.8.4 Genetic Algorithms

A genetic algorithm is a smart-and-lucky search that manipulates a population of points in the effort to find the optimal solution. Each individual in the population is a string of genes, where each gene describes some feature of the solution. Genetic algorithms mimic the processes of natural evolution, including reproduction and mutation. The most powerful operator of a genetic algorithm is the crossover operator: the recombination of the genes of two parents to

create two offspring. This crossover allows the offspring to acquire the good characteristics of different points in the search space.

Most genetic algorithms perform the following steps, stopping when a fixed number of offspring has been created:

Step 0: Form an initial population.
Step 1: Evaluate the individuals in the population.
Step 2: Select individuals to become parents with probability based on their fitness.
Step 3: For each pair of parents, perform a crossover to form two offspring. Mutate each offspring with some small probability.
Step 4: Place the offspring into the population. Return to Step 1.

Holland (1975), Davis (1987, 1991), and Goldberg (1989) provide good descriptions of genetic algorithms. Holland's 1975 book introduced the genetic algorithm (GA), a procedure that mimics the adaptation that nature uses to find an optimal state. In genetic algorithms, solutions are represented as strings (chromosomes) of alleles, and the search performs operations on the population of solutions. Liepins and Hilliard (1989) identify these operations as 1) the evaluation of individual fitness, 2) the formation of a gene pool, and 3) the recombination and mutation of genes to form a new population. After a period of time, good strings dominate the population, providing an optimal (or near-optimal) solution.

The solution strings may be a sequence of binary bits or a permutation. The fitness of a solution is its objective function evaluation or ranking. In forming the gene pool, the algorithm takes, through some random process, those solutions that are more fit. The recombination is some type of crossover, one-point or multipoint. It is this powerful crossover mechanism that makes GAs special. The mutation operation, so important in SA, is a secondary operation here and serves only to maintain diversity.

Why do GAs work? A schemata is a pattern possessed by individuals in population. The fitness of a schema is the average fitness of those solutions that possess it. Genetic algorithms work because good (short) schema survive exponentially, and the population of solutions provides implicit parallelism.

When do they fail? A genetic algorithm may fail if the strings are inappropriate representations of solutions, the strings exclude important problem information (such as constraints), or the algorithm converges to a local optimum. The first issue is the largest for scheduling problems: representation is difficult for combinatorial problems because standard crossovers may lead to infeasibility.

In an effort to address this problem for the traveling salesman problem, where a solution is a permutation of cities, Oliver, Smith, and Holland (1987) propose a number of unusual crossovers. The first is the Order crossover, which substitutes a subsequence from the first parent into the second and then visits the remaining cities in the same relative order as before. The crossover maintains short schema. Their PMX crossover compares subsequences from both parents and then performs a swapping in the second parent. The Cycle crossover matches cities to form independent tours. Then, for each tour, the crossover picks a parent to supply the cities. After comparing the tree crossovers, they claim that the Order is better on TSP by preserving the short schema, which are important for this problem.

The job shop scheduling problem is much more complicated problem. Davis (1985) solves a simple job shop problem with a GA. Another method is an idea by Storer, Wu, and Vaccari (1990, 1992): searching problem spaces and heuristic spaces, which are discussed in detail in Section 2.9.

Storer, Wu, and Vaccari (1990) note that a solution is simply the application of a heuristic to a problem. Changing the problem or the heuristic generates a new schedule and thus a new solution. A problem is a vector of processing times, and a heuristic is a vector of dispatching rules that can be used to create a non-delay or active schedule. These data structures create simple strings and perform well under standard crossover as well as simulated annealing and tabu search. The authors perform all three searches on both types of spaces. For the tabu search, the tabu list was a set of tabu makespans. Otherwise, the SA and TS were standard. The authors also tried shifting bottleneck, probabilistic dispatching, and random search over their new spaces. They test their heuristics on a number of problems ranging in size from 10 to 50 jobs. The

problems were also classified as easy or hard. The easy problems had completely random routings, and the hard problems tended have greater competition for the machine resources at any point in time. The genetic algorithms on problem space generally found higher-quality solutions.

Fox and McMahon (1990) are interested in the job shop scheduling problem but study the traveling salesman problem in order to gain insight into using genetic algorithms for sequencing problems.

The authors consider the binary precedence matrix M where $m_{ij} = 1$ if city i precedes city j. $m_{ii} = 0$ for all cities i. This matrix incorporates micro- and macro-information. The authors consider row and column swap operators that exchange the successors or predecessors of two cities.

The first new operator is the intersection operator, in which an offspring inherits the precedences that exist in both parents. That is $c_{ij} = a_{ij}$ and b_{ij}, over all i,j. This will yield a matrix that is consistent (no cycles) but incomplete; in other words, a partial ordering. The matrix is completed through an analysis of the row and column sums.

The second new operator is the union operator. First, the cities are partitioned into two sets. This forms for each parent two precedence submatrices corresponding to the two subsets. The operator then takes one submatrix from one parent and the submatrix for the other subset from the other parent. Again, this leads to a partial ordering that must be completed. The authors claim that this operator contains little micro-information from the parents, unless a Markov process is used to partition the cities.

The authors compare different operators for the TSP, including a random operator used as a benchmark. If the proposed operators are no better than this, they should not be pursued. Their city topologies included random distances, clustering, concentric circles, and a 30-city problem from Oliver, Smith, and Holland (1987). The population size was 900 with no mutation. The also tried searches with and without elitism (the keeping of the best solution).

Among non-elite searches, the two new operators were great, even when considering processing time. Their advantage disappeared when the searches got to use elitism.

The authors claim that this work will be useful in scheduling problems, although execution time remains a big problem.

Biegel and Davern (1990) begin with a general discussion of genetic algorithms. They then describe how a genetic algorithm could play a part in the scheduling of a shop. They discuss what data are essential and how the GA could form a schedule for a known group of jobs that follow the same route.

The authors move to the single-machine problem and consider how the analogous GA would work. Moving to the 2- and *m*-machine flow shop, the authors present a simple discussion of what the GA would do.

For the dynamic problem, the authors consider performing a rescheduling in real-time to deal with arrivals and use a FIFO dispatching rules on all other machines. The authors then list a number of areas for future research.

Cleveland and Smith (1989) use a genetic algorithm to schedule the release of jobs into a manufacturing cell to minimize weighted tardiness. They consider both the sequencing problem and the problem of determining the release times.

Nakano and Yamada (1991) introduce a binary representation to solve the job shop scheduling problem. This representation denotes the relative ordering of each pair of jobs on each machine. If an illegal chromosome is formed by a genetic operator, a nearby legal one replaces it. The authors report that their algorithm finds good solutions on some standard problems.

Whitley, Starkweather, and Shaner (1991) develop an edge recombination operator for the traveling salesman problem and report that a genetic algorithm with this operator is able to find very good schedules.

Starkweather *et al.* (1991) compare six sequencing operators for the traveling salesman problem. They discover that the different operators stress different types of information; since the TSP depends upon adjacency, the edge recombination operator is the best.

In Syswerda (1991), the author discusses the scheduling of a fighter simulation lab. Finding a feasible scheduling is made difficult by a number of constraints. A genetic algorithm that manipulates sequences of jobs and employs a schedule builder to translate the sequence into a legal schedule was able to find good schedules.

2.8.5 Summary

Simulated annealing and tabu search easily search complex spaces by lending themselves to usually simple types of moves, generally find good solutions fast, and are smart variations of standard searches such as hillclimbing and steepest descent. Genetic algorithms work in a different manner. They work well due to the survival of good schema and their implicit parallelism. They have been harder to implement on scheduling problems, however.

This research in this dissertation builds upon this work into smart-and-lucky searches by looking into some alternative search spaces.

2.9 Problem and Heuristic Space

This dissertation includes the development of global job shop scheduling models, whose solution will yield a schedule that can be followed for time period like an eight-hour shift. As mentioned in the last section, one way to find good schedules is to use a search. One recently-proposed idea is to search problem and heuristic spaces. This section will define mathematically how these spaces can be used and will review the ideas of a few papers in this area.

Define a *problem* p as a set of data about which an optimization question can be asked. A *solution* s is a point that is consistent with the problem structure, where the solution has some performance z measured by applying the objective function f to the solution with the problem data, $z = f(p,s)$. If the problem is fixed, there is a performance function C_p over the solution space such that $z = C_p(s) = f(p,s)$. Solving a problem translates as finding the solution that gives the minimum function value.

Traditionally, problem-solving searches have occurred in the solution space. In the solution space live points (for standard optimization) or sequences (for combinatorial optimization). Heuristics are also applied to problems. A *heuristic h*, applied to a problem p, yields a solution s, $s = G(h,p)$ or $s = h(p)$. Thus, for a given problem p, there exists a performance function $D_p(h)$ over the heuristic space such that $z = D_p(h) = C_p(h(p))$. This implies that a search over different heuristics could find a solution that gives the optimal objective function value.

Moreover, a solution can be generated in ways besides applying a heuristic to the original problem. Applying a heuristic h to a problem y in another space can generate a solution $s = h(y)$, which can be evaluated using the objective function. Thus, given a problem p and the heuristic h, there exists a performance function $E_{p,h}(y)$ over the other problem space, where $z = E_{p,h}(y) = C_p(h(y))$. And as before, a search over this new problem space provides a way to solve the problem.

The idea of searching heuristic and problems spaces was investigated by Storer, Wu, and Vaccari (1990, 1992). In these papers, the authors are investigating the general job shop scheduling problem. They define a heuristic space composed of vectors of dispatching rules. The heuristic uses each rule in turn for a fixed number of dispatching decisions. For example, if the problem is a ten-job by ten-machine problem, the vector might have five elements, where the dispatching rule in the first element is used for the first window of twenty decisions, the next rule for the next window, and so on, until all one hundred operations have been processed.

For a problem space, they use a space of vectors of processing times, where each position corresponds to a different operation. At each scheduling decision in the formation of the schedule, the dispatching rule selects a job by considering these alternative processing times. The operation of that job is scheduled using the actual job data in order to calculate the finish time of the operation, thus ensuring that the schedule produced will be feasible. Thus, a vector of processing times yields a sequence of operations for each machine.

The authors include a number of concepts related to searching these new spaces, including the definitions of neighborhoods, the use of these spaces in genetic algorithms, the fact that the spaces contain optimal solutions, and details of local search implementations.

The application to genetic algorithms merits special comment. Traditional genetic algorithms have difficulty with scheduling problems since the components of the strings (the jobs in the sequence) are not independent of each other. The problem space and heuristic space described above, however, consist of vectors that have independent elements. That is, the dispatching rule (or processing time) in the first element does not affect what values the other elements can have. Thus, a crossover operation that breaks two strings (vectors) and joins the separate pieces yields offspring that are valid points in the search space.

The authors extend this work in Wu, Storer, and Chang (1993) to the one-machine rescheduling problem. They use heuristics that adjust "artificial tails" to find schedules that are efficient and deviate little from the original schedule.

Bean (1992) uses the idea of random keys to provide an alternative search space for a number of problems: multiple-machine scheduling, resource allocation, and quadratic assignment: the keys are used to sequence the jobs (or other variables). Then some simple rule is used to generate a solution from this sequence. Good empirical results are reported.

Our research extends these ideas to one-machine class scheduling problems and investigates a similar heuristic space for the job shop scheduling problem.

2.10 NP-Completeness

Job shop scheduling problems are usually quite difficult to solve, and they are among the hardest optimization problems known. Combinatorial problems, as well as other types of problems, can be classified by their complexity, or how difficult they are to solve. The hardest problems are called NP-complete. The primary guide to the theory and use of NP-completeness is the book by Garey and Johnson (1979).

Most of the problems studied in operations research have been classified by their complexity, and as researchers study new problems, their complexity is identified also. There are three primary classifications: polynomial, NP-complete, and strongly NP-complete. Problems in the first set can be solved with algorithms that have a running time proportional to some polynomial function of the problem size. Algorithms to solve problems in the second set require effort that is a polynomial function of the problem size and the problem data. Such an algorithm is called a *pseudo-polynomial* algorithm. Algorithms to solve problems that are strongly NP-complete require an exponential amount of effort.

The complexity of a problem can be determined by identifying the problem as a more difficult case of a hard problem, by developing a polynomial algorithm to solve the problem, or by transforming a previously classified problem into the new problem. By being able to determine the complexity of a problem, researchers are able to know what types of approaches may be possible in finding optimal solutions to the problem. A problem that is known to be NP-complete will not be solved optimally by any polynomial-time algorithm. Pseudo-polynomial dynamic programs will not solve strongly NP-complete problems. This does leave the possibility, however, that certain special cases may be more easily solved, that algorithms that search for an optimal solution may have good average performance (e.g. the simplex method of solving linear programs), and that polynomial-time heuristic algorithms may be able to find generally high-quality solutions.

This research is concerned with a number of different scheduling problems, which will classified by their complexity as necessary. We will directly prove NP-completeness for one of the problems that we study. Each of the other problems is a more difficult case of a previously considered NP-complete problem.

2.11 Chapter Summary

This chapter has attempted to cover a broad spectrum of topics and literature. Obviously, much research has been done in the fields of job and flow shop scheduling and the field of

managing semiconductor manufacturing. However, no system to optimize the scheduling of such a process has emerged, despite the use of many different and sophisticated procedures.

Still, new tools are needed that may have good performance when applied to specific settings. There do exist new ideas, such as class scheduling, look-ahead and look-behind approaches, smart-and-lucky searches, and problem and heuristic spaces. The literature in these areas reports on some initial research.

However, the full capabilities of the these methods, especially when joined to solve a scheduling problem, have not been fully explored.

CHAPTER 3
ONE-MACHINE CLASS SCHEDULING PROBLEMS

In Chapters 3, 4, and 5, we discuss the results of our research into a number of different scheduling problems and the general job shop scheduling problem. The primary motivation is to examine subproblems of the job shop scheduling problems in order to gain insight into the larger problem. The subproblems that we will examine are interesting scheduling problems that have not been previously considered.

We will start with the one-machine class scheduling problems. Our approach in this chapter is to develop analytical results, to test extended heuristics, and to show that a problem space genetic algorithm can be a good procedure for a variety of scheduling problems.

3.1 Introduction

The three one-machine problems studied in this work are class scheduling problems that model the complicating factor of machine setups in the manufacturing process. Most problems with sequence-dependent setup times are NP-complete. Class scheduling problems have a special structure that makes them good candidates for further research: the jobs to be scheduled form a number of disjoint job classes and setups occur whenever the machine processes consecutive jobs from different classes. And although we will study a number of different problems, we will see that a problem space genetic algorithm will be a useful procedure for all of them.

The one-machine class scheduling problems under investigation are as follows:

1. Constrained Flowtime with Setups (CFTS)
2. Class Scheduling with Release and Due Dates (CSRDD)
3. Flowtime with Setups and Release Dates (FTSRD)

The first problem studied is the class scheduling extension of the one-machine problem of minimizing the total flowtime of a set of jobs that have deadlines. The research includes an optimality property for the jobs in the same class, a good heuristic, and the development of a problem space genetic algorithm that can find better solutions.

The second problem is the class scheduling problem where each job has a release date and a due date. The objective is to minimize the number of tardy jobs. We investigate a number of heuristics and the use of a problem space genetic algorithm. We also look at a secondary criteria, minimizing the total tardiness.

The objective in the third problem is to minimize the total flowtime where each job has a release date. We consider a number of approaches to finding good solutions, including a problem space genetic algorithm, and present a special case that can be solved with a pseudo-polynomial dynamic programming algorithm.

This chapter considers the research on each of these problems in turn. Research relevant to these problems is also discussed in Chapter 2. See especially Sections 2.6 and 2.7.

3.2 Constrained Flowtime with Setups

We will first consider the one-machine class scheduling problem of minimizing the total flowtime subject to the constraint that each job must finish before its deadline. A new heuristic is proposed for the problem. We investigate the use of a genetic algorithm to improve solution quality by adjusting the inputs of the heuristic. We present experimental results that show that the use of such a search can be a successful technique.

3.2.1 Introduction

This research is motivated by the scheduling of semiconductor test operations. Assembled semiconductor devices must undergo electrical testing on machines that can test a number of different types of semiconductors. If a machine is scheduled to test a lot consisting of devices that are different from the devices tested in the previous lot, various setup tasks are required.

These tasks may include changing a handler and load board that can process only certain types of semiconductor packages or loading a new test program for the new part. However, if the new lot consists of circuits that are the same as the previous type, none of this setup is required. This type of change is a sequence-dependent setup that can be modelled by class scheduling.

Since post-assembly testing is the last stage in semiconductor manufacturing, meeting a job's due date is a very important objective for the manager of a test facility. A secondary criterion is the minimization of total flowtime (the sum of the job completion times), which reflects the manager's desire to increase throughput and decrease inventory holding costs.

This is a dual criteria problem, in which the primary criterion is used as a constraint and the secondary criterion is optimized under this restriction. The problem of minimizing the total flowtime subject to deadlines is an old problem. Smith (1956) provides an optimal solution technique that repeatedly schedules the longest eligible job last. The class scheduling version of the problem, however, is more difficult.

For our problem, finding a feasible schedule is an NP-complete problem. (A schedule is feasible if every job finishes before or at its deadline.) Thus, there exist no exact algorithms to minimize in polynomial time the total flowtime subject to the deadline constraints. (For a discussion of the theory of NP-completeness, see Section 2.10 and Garey and Johnson, 1979.) Thus, we are motivated to try different heuristics. In this work we develop a multiple-pass heuristic that finds good solutions quickly. The first contribution of our investigation of this problem is the extension of Smith's algorithm into a heuristic which considers the setup times while sequencing the jobs by their deadlines and processing times.

We are also interested in using a genetic algorithm to improve the quality of our solutions. A genetic algorithm is a heuristic search that has been used to find good solutions to a number of different optimization problems, but genetic algorithms searching for good schedules must overcome the difficulty of manipulating the sequences of jobs. We investigate the use of a genetic algorithm to search a new type of space, the problem space. This type of approach was introduced in Storer, Wu, and Vaccari (1992), who consider alternative search spaces for the

general job shop scheduling problem. In this work, we extend the idea to the problem of one-machine class scheduling.

Our search attempts to adjust the deadlines of the given problem so that our heuristic will find even better solutions. The space of adjusted deadlines that we search forms a *problem space* (we will return to this point in Section 3.2.5). The second contribution of this work is our use of this method to improve the scheduling of a single-machine problem.

If we use the principles of Davis (1991), then we can classify our genetic algorithm as a type of hybrid genetic algorithm. However, the only unusual characteristic of our algorithm is the decoding (the bit string does not describe a point in the solution space; instead it must be mapped to a solution via the heuristic). Moreover, the range of hybrid genetic algorithms is so large (for instance, Goldberg, 1989, describes hybrids differently) that our use of the term *problem space genetic algorithm* is a more precise description of the search. Finally, this problem space exists independently of the genetic algorithm, and the use of this new search space is not limited to our search. Other searches (including steepest descent, simulated annealing, and tabu search) could be used to explore the space. Therefore, we will continue to refer to our search space as a problem space and to our search as a problem space genetic algorithm.

The next subsection summarizes some of the relevant literature on class scheduling problems and the dual criteria objective under consideration. In Section 3.2.3, we discuss our notation, an example instance of the problem, and a number of basic results. We discuss in Section 3.2.4 the heuristic developed for the problem. Our genetic algorithm will employ this heuristic. In Section 3.2.5, we present a problem space, introduce genetic algorithms, and discuss the details of the genetic algorithm we developed to search the problem space. Section 3.2.6 describes the generation of the sample problems, the computational experiments, and the results. Finally, in Section 3.2.7, we present our conclusions.

3.2.2 Literature Review

In this section we will briefly mention some of the relevant research on class scheduling and on the dual criteria problem of minimizing total flowtime subject to job deadlines. This owrk and the literature on genetic algorithms are discussed in more detail in Chapter 2.

One of the first papers on problems with class scheduling characteristics is Sahney (1972), who considers the problem of scheduling one worker to operate two machines in order to minimize the flowtime of jobs that need processing on one of the two machines. Sahney derives a number of optimal properties and uses these to derive an branch-and-bound algorithm for the problem. Gupta (1984) defines the class scheduling problem, and Potts (1991), Coffman, Nozari, and Yannakakis (1989), and Ho (1992) also study two-class scheduling problems.

Bruno and Downey (1978) prove that, for more general class scheduling problems, the question of finding a schedule with no tardy jobs is NP-complete. Monma and Potts (1989) prove that many class scheduling problems are NP-complete, including minimizing makespan, maximum lateness, the number of tardy jobs, total flowtime, and weighted flowtime.

Dobson, Karmarkar, and Rummel (1987, 1989), Gupta (1988), Ahn and Hyun (1990), and Mason and Anderson (1991) all study the class scheduling problem under different objective functions. We will modify the procedure of Ahn and Hyun in order to use it as a comparative heuristic. The only other dual criteria problem in this area is studied by Woodruff and Spearman (1992); they consider a class scheduling problem with profit maximization and deadlines.

In the dual criteria literature, the problem of minimizing total flowtime subject to job deadlines (a deadline is a constraint on the completion time) is among the oldest questions, being first studied by Smith (1956). The problem of minimizing the weighted flowtime subject to job deadlines is a strongly NP-complete problem (Lenstra, Rinnooy Kan, and Brucker, 1977), and a number of researchers have examined branch-and-bound techniques.

3.2.3 Notation and an Optimal Property

In this section we introduce our problem and notation, give an example instance of the class scheduling problem under consideration, and present some basic results.

Our class scheduling problem is the minimization of the total flowtime of a set of jobs where the jobs have deadlines on their completion. The problem can be formulated with the following notation:

J_j	Job j, $j = 1, \ldots, n$
p_j	processing time of J_j
D_j	deadline of J_j
G_i	job class i, $i = 1, \ldots, m$
n_i	number of jobs in G_i
s_{0i}	time of initial setup if first job is in G_i
s_{ki}	time of setup between jobs in G_k and G_i
C_j	completion time of J_j
$\sum C_j$	total flowtime.

The problem is to find a sequence that minimizes the total flowtime ($\sum C_j$) subject to the deadline constraints ($C_j \leq D_j$ for all J_j). We name this problem the *Constrained Flowtime with Setups* problem (*CFTS*). Since job preemption or inserted idle time leads to a non-optimal solution, we will assume that schedules being considered have neither. Any schedule that is a solution for CFTS will have a number of *batches* or *runs* that are sets of jobs from one class processed consecutively. Before each batch will be a class setup. The problem involves determining the composition and order of batches from different classes.

CFTS is an extension of a one-machine problem studied by Smith (1956). In his problem, which we name the *Constrained Flowtime* problem (*CFT*), there exist no sequence-dependent setup times.

An instance that we will use to illustrate our work is described in Example 3.1.

Example 3.1. The data in Table 3.1 form an instance of a class scheduling problem with five jobs in two job classes. The first three jobs form one class, with the remaining two jobs in the

second class. Recall that no setup is required between jobs in the same class. However, a class setup is necessary between jobs of different classes.

Table 3.1. Job and Class Data for Example 3.1.

<table>
<tr><td>j</td><td colspan="5">1   2   3   4   5</td></tr>
<tr><td>p_j</td><td colspan="5">1   2   2   3   2</td></tr>
<tr><td>D_j</td><td colspan="5">3   16   14   10   18</td></tr>
<tr><td colspan="2">$G_1 = \{1, 2, 3\}$</td><td colspan="2">$n_1 = 3$</td><td colspan="2">$s_{01} = s_{21} = 2$</td></tr>
<tr><td colspan="2">$G_2 = \{4, 5\}$</td><td colspan="2">$n_2 = 2$</td><td colspan="2">$s_{02} = s_{12} = 1$</td></tr>
</table>

CFTS is an NP-complete problem since finding a feasible schedule is NP-complete (Bruno and Downey, 1978). Hence, it is unlikely that any polynomial algorithm to solve the problem exists. We will study the use of heuristics to find good solutions.

We now describe Smith's rule for CFT and an optimal property for CFTS. Our new heuristic extends Smith's rule by taking advantage of the optimal property, which we call Smith's property.

In this and later sections, we will refer to each job that can complete at a given time without violating the job's deadline as being *eligible*. In this problem, we are concerned with deadlines that are constraints on the completion times, and we will create schedules backwards, starting with the last position in the sequence. Thus, we say that a job J_j is eligible at a time t if $t \leq D_j$. The job can feasibly complete at this time without violating the deadline constraint.

Smith's property for CFT states that if a job is assigned the last position of an optimal schedule, then it must be the longest eligible job. Smith's rule is derived from this property.

Algorithm 3.1 (Smith's rule for CFT). Let $t = p_1 + \ldots + p_n$. Among the jobs that are eligible at time t, that is, $t \leq D_j$, choose the job J_j with the longest processing time p_j. Schedule J_j to complete at t, and solve the remaining problem in a similar manner.

The following property extends Smith's rule to each class in CFTS. This property will then be extended to consider all classes in order to generate approximate solutions to CFTS.

Lemma 3.1 (Smith's property for CFTS). For each class in an optimal schedule for CFTS, the only job that could be scheduled to complete at a time t is the longest eligible one.

Proof. It suffices to show that if two jobs J_i and J_j in the same class are both eligible at time t and J_i is longer than J_j ($p_i > p_j$), then scheduling J_j to complete at time t leads to a non-optimal solution. Suppose we do. Then $C_j = t$, and J_i precedes J_j in the schedule formed. Create a new schedule by interchanging the two jobs. Since J_j is moved to the left, it is still feasible, and the new completion time is less than C_i, the old completion time of J_i ($p_i > p_j$). The completion times of any jobs between J_j and J_i are decreased. Meanwhile, J_i completes when J_j did, but this is feasible since $t \leq D_i$. We have therefore created a feasible schedule with less total flowtime, and the original schedule cannot be optimal. QED.

3.2.4 The Heuristic

Quick methods of finding good solutions are sometimes effective ways to attack difficult problems. In this section we describe a multiple-pass heuristic that extends the idea of Smith's rule. We illustrate how this heuristic works using Example 3.1.

Our heuristic finds solutions for CFTS by scheduling jobs in the spirit of Smith's rule, working backwards from the end of the schedule. Since the makespan (the maximum completion time) of the optimal solution is not known, the heuristic starts with a trial makespan. After scheduling all of the jobs, we compute the actual makespan (by removing any idle time) and use this makespan as the starting point for another iteration. We continue this process until some limiting makespan is reached. At this point, another pass of the heuristic yields a schedule with the same makespan or a schedule that is infeasible (because some job or setup starts before time zero).

This heuristic constructs schedules that satisfy Smith's property for CFTS (Lemma 3.1). While that lemma applies only to jobs in one class, our algorithm extends the idea of longest

eligible job by considering all of the job classes. We schedule the longest job with the minimum wasted time. Wasted time is time spent in a setup and idle time.

This (single-pass) Minimum Waste algorithm schedules an eligible job from the same class as the previously scheduled job if one exists. Else, it selects a job from the class with the smallest setup or selects the job with the latest deadline. It does this by measuring each job's *gap*: the wasted time incurred by selecting that job. Note that if no class setups exist, this algorithm is the same as Smith's rule for CFT (Algorithm 3.1).

Algorithm 3.2 (Single-pass) Minimum Waste.

Step 0: Given a completion time t, select for the last job the longest job eligible at this time J_j, i.e. $t \leq D_j$. Schedule this job to end at t, and reduce t by p_j.

Step 1 (a modification to Smith's rule): Suppose that at time t, a job from class G_i starts. Then, for each unscheduled job J_j, define q_j as the gap between the last possible completion time of J_j and t. If J_j is in class G_k, $q_j = \max\{t - D_j, s_{ki}\}$ (see Remarks below for an explanation of this definition). Let $q = \min\{q_j \text{ over unscheduled } J_j\}$. Select the longest job J_j with $q_j = q$ and schedule this job to end at $t - q$. Any necessary setup s_{ki} can begin at $t - q$. Reduce t by q and p_j.

Step 2: If there remain unscheduled jobs, return to Step 1.

Step 3: There are no more unscheduled jobs. If the first job in the schedule is in class G_i, a setup of length s_{0i} must end at t. Reduce t by this amount.

Step 4: If $t < 0$, the schedule created is infeasible. Else, compute the actual makespan of the jobs and setups scheduled.

Remarks. In order to motivate the definition of q_j, the gap, let us note that the setup after J_j is s_{ki}, so the job may not complete after $t - s_{ki}$. However, if the deadline $D_j < t - s_{ki}$, the gap will be $q_j = t - D_j$, which is greater than s_{ki}. This gap thus includes the setup and a period of inserted idle time of length $t - D_j - s_{ki}$. Note that if J_j is also in class G_i, $q_j = 0$ if and only if $t \leq D_j$. The algorithm is a type of greedy heuristic, in that it attempts to minimize the setup time or idle time in selecting jobs to be scheduled.

Multiple-Pass Minimum Waste Heuristic. To find a good solution for CFTS, we can use the following procedure that makes use of the single-pass Minimum Waste algorithm.

Step A: Let $t' = \max \{D_j : j = 1, \ldots, n\}$.

Step B: Let $t = t'$.

Step C: Perform one pass of the Minimum Waste algorithm (Algorithm 3.2) with completion time t. This creates a trial schedule.

Step D: Let t' be the sum of processing times and setup times of this schedule. If $t' < t$ and the trial schedule is feasible, go to Step B. (The smaller makespan may yield another schedule.)

Step E: If the trial schedule was infeasible or $t' = t$, take the last feasible schedule created and remove the inserted idle time, starting all jobs as soon as possible. This schedule is the result of the heuristic. If an infeasible schedule was created on the first pass, then take the sequence of jobs from the schedule and process the jobs in this order, starting at time zero. This will yield a schedule with some violated deadline constraints.

Because the problem of finding a feasible schedule is NP-complete, a single pass of the Minimum Waste algorithm is not guaranteed to find one. Still, as we shall see, it is usually able to find a feasible schedule if one exists. If a feasible schedule exists, it must finish by the maximum deadline, which is the first trial makespan. Initially, the heuristic is concerned with reducing the makespan. Eventually, as the makespan reaches a lower limit, the algorithm concentrates on the flowtime objective through its use of Smith's property to select a job.

Example 3.2. Let us apply the Multiple-Pass Minimum Waste heuristic to Example 3.1. In the first iteration, $t = 18$, the maximum deadline. The first pass of the Minimum Waste algorithm performs the following calculations (see Table 3.2 for complete algorithm): at time 18, no jobs have been scheduled, and the only eligible job is J_5. After choosing J_5, t is reduced by $p_5 = 2$ to 16. For J_1, J_3, and J_4, the waste is the gap until the deadline. Thus, $q_1 = 16 - D_1 = 13$, and similarly for the other two jobs. For J_2, however, $D_2 = 16$, and the deadline gap is zero, but because J_2 is in a different class than J_5, $q_2 = s_{12} = 1$. Thus, J_2 has the smallest waste and is scheduled to end at time 15. After five steps, all of the jobs are scheduled (see Figure 3.1). There

are two units of inserted idle time, however, so the actual makespan of the schedule can be reduced to 16.

Table 3.2. Calculations of the first pass of the Minimum Waste algorithm. Initial makespan = 18.

	Waste:					Schedule:	
Time:	J_1	J_2	J_3	J_4	J_5	J_j	C_j
18	15	2	4	8	0	J_5	18
16	13	1	2	6		J_2	15
13	10		0	3		J_3	13
11	8			2		J_4	9
6	3					J_1	3
Scheduled flowtime: 58. Reduced makespan: 16.							

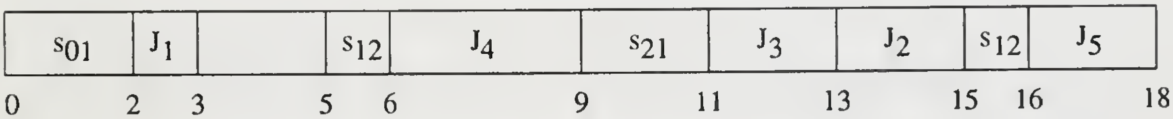


Figure 3.1. Schedule created on first pass of heuristic.

When the heuristic repeats the algorithm with the new makespan of 16 (see Table 3.3 for calculations), jobs J_2 and J_5 are eligible at time 16. These jobs also have the same processing time, but suppose J_2 is chosen. Then, at time 14, J_3 is eligible and has no gap, as it is in the same class as J_2. However, J_5 has a gap $q_5 = s_{12} = 2$. The algorithm continues in this manner, creating the schedule shown in Figure 3.2.

The Multiple-Pass Minimum Waste Heuristic again repeats the algorithm, which now begins at the reduced makespan of 15, and a similar schedule can be found if J_2 is chosen at time 15. This schedule has no idle time (see Figure 3.3), the reduced makespan is also 15, and so the heuristic stops.

Table 3.3. The second pass of the Minimum Waste algorithm. Initial makespan = 16.

Time:	Waste: J_1	J_2	J_3	J_4	J_5	Schedule: J_j	C_j
16	13	0	2	6	0	J_2	16
14	11		0	4	2	J_3	14
12	9			2	2	J_4	10
7	6				0	J_5	7
5	2					J_1	3
Scheduled flowtime: 50. Reduced makespan: 15.							

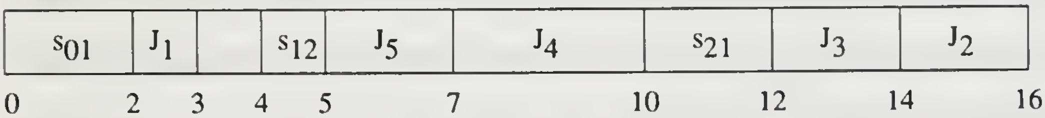


Figure 3.2. Second schedule.

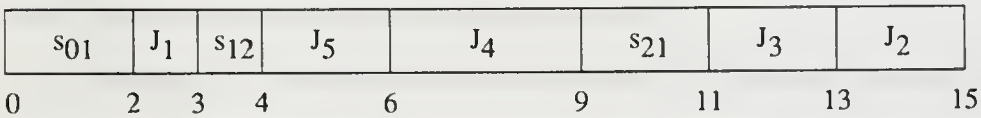


Figure 3.3. Third and final schedule.

3.2.5 The Genetic Algorithm

In this section we present an alternative search space, the problem space, provide a brief introduction to genetic algorithms and some references to selected works, and discuss the details of the problem space genetic algorithm we used to find good solutions for CFTS.

Problem space. The original work on problem and heuristic spaces is by Storer, Wu, and Vaccari (1990, 1992), who define some alternative search spaces for the job shop scheduling problem: the *problem space* and the *heuristic space*. They note that a solution to a problem is the result of applying a heuristic to the problem. Given a problem p, a heuristic h is a function that creates a sequence corresponding to a solution s, i.e. $h(p) = s$. Thus, if one adjusts the heuristic, one creates a different solution. The set of adjusted heuristics is the heuristic space. Likewise, if

one adjusts the problem data that are used by the heuristic, one generates a different solution. This set of adjusted problem data is the problem space. The idea is applied to the job shop scheduling problem, and different heuristic searches over the spaces are performed, including hill climbing, genetic algorithms, simulated annealing, and tabu search.

Our research extends this idea by defining a problem space for the one-machine class scheduling problem. If we adjust the deadlines that are inputs (along with the other problem data) to a pass of the Minimum Waste algorithm, we will create a possibly different schedule. We will use as a problem space for CFTS these *adjusted deadlines*, and we will use one pass of the Minimum Waste algorithm to create a sequence of jobs using the adjusted deadlines. The feasibility (against the actual deadlines) and total flowtime of the sequence can be evaluated by scheduling the jobs to start at time zero with no inserted idle time. The idea is to force jobs to be done earlier or later by decreasing or increasing the deadlines. We will prove that every solution for CFTS (including the optimal one) is in the range of h.

Theorem 3.1. For each solution to an instance of CFTS, there exists a vector of adjusted deadlines that can be mapped to that solution using one pass of the Minimum Waste algorithm.

Proof. Suppose that σ is a solution (a feasible schedule with no preemption or inserted idle time) for an instance of CFTS. For each job, consider adjusting the deadline so that it equals the job completion time. Then, if we use one pass of the Minimum Waste algorithm with the adjusted deadlines, the job selected for the last position will be the job with the maximum adjusted deadline. This job is the one with the maximum completion time C_i and thus was the last job in σ. It will be scheduled to complete at its adjusted deadline, which is C_i.

Now we are at the start time t of a job J_i and the job with the smallest gap is the unscheduled job J_j that immediately precedes J_i in σ, since the adjusted deadline is C_j, and $C_j \leq t$. Any setup necessary between J_j and J_i is already included in the difference between C_j and t. Thus the gap cannot be larger for this job, and the gap for any other job J_k is larger since $C_k < C_j$. This job will be scheduled to complete at its adjusted deadline, which is C_j. If we continue in this

manner, all of the jobs will be sequenced in the same order as they were in σ, and we create the same schedule. QED.

The more we adjust the deadlines, the more change we create in the schedule. For instance, consider the following examples of applying one pass of the Minimum Waste algorithm to vectors of adjusted deadlines where we have changed only the second and fifth deadlines: (Note that the fourth schedule created is infeasible since J_2 completes at time 18, which is greater than the actual deadline: $D_2 = 16$. The adjusted deadline of 19 was used only to sequence the jobs.)

Heuristic(problem) = solution:

Minimum Waste (3, 6, 14, 10, 20) = [1 2 4 3 5], flowtime 46.
Minimum Waste (3, 16, 14, 10, 18) = [1 4 3 2 5], flowtime 50 (original deadlines).
Minimum Waste (3, 17, 14, 10, 16) = [1 5 4 3 2], flowtime 46.
Minimum Waste (3, 19, 14, 10, 17) = [1 4 3 5 2], infeasible ($C_2 = 18$).

In Figure 3.4 we show a graph that illustrates how adjusting just two of the five deadlines of Example 3.1 can create a number of different schedules. The first, third, and fourth deadlines were not adjusted. Each point in the plane (only non-negative deadlines were considered) corresponds to a pair of values for the second and fifth adjusted deadlines. The points in each region of the plane are mapped by a pass of the Minimum Waste algorithm to the job sequence denoted by the five-digit sequence shown in that region. The dot marks the point that corresponds to the unadjusted deadlines ($D_2 = 16$, $D_5 = 18$). The best sequences achievable by adjusting these deadlines are 12435 and 15432 (total flowtime = 46), and the only other feasible sequences are 14235 and 14325 (total flowtime = 50). The optimal solution (which cannot be found by adjusting only the second and fifth deadlines) is 13452, with total flowtime = 43.

Since the actual problem space consists of all of the problem data and there are numerous heuristics that can be used, we can investigate other spaces and heuristics that might be useful. Our first search was to adjust the job processing times and to use the Shortest Processing Time (SPT) rule. However, it is difficult to find feasible solutions since SPT ignores the deadline constraints entirely. We also tried the using the Earliest Due Date (EDD) rule while adjusting the

deadlines, but EDD does not give enough attention to the flowtime objective. Sequencing by either SPT or EDD is a fairly naive heuristic, since neither makes use of the other available information. The Minimum Waste algorithm, however, considers due dates, processing times, and setups, and using it improves our searches. In addition, while it would be possible to use the Minimum Waste algorithm while adjusting the processing or setup times, the effect of these variables on the sequencing of jobs is more indirect than that of the deadlines.

The use of a heuristic space seems to be hard for this problem. Feasibility is a large concern, and there are very few heuristics we can use to find feasible schedules. Also, the Minimum Waste algorithm has no parameters to adjust.

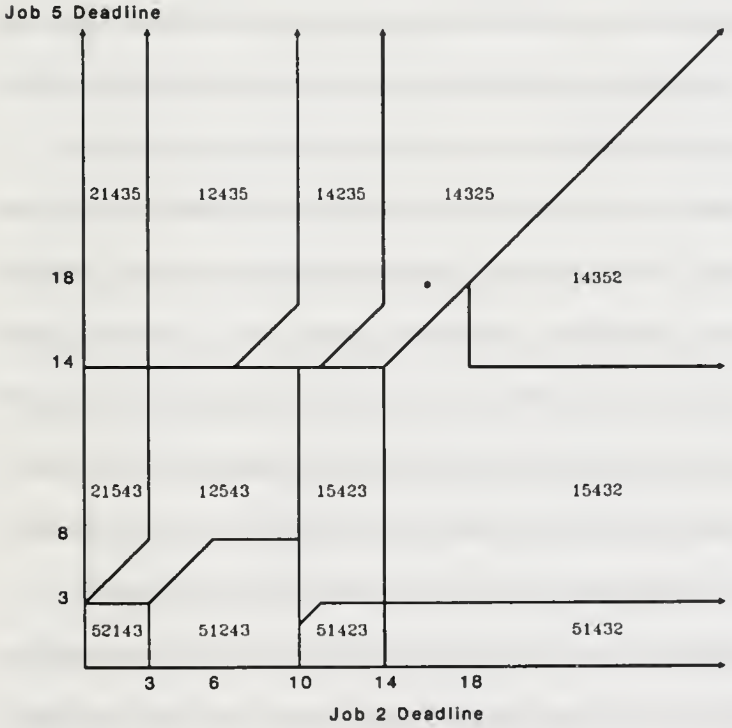


Figure 3.4. Graph of adjusted deadlines and schedules created.

A genetic algorithm for CFTS. In this section we discuss the details of the genetic algorithm we developed to find better solutions for CFTS. As mentioned before, genetic algorithms are heuristic searches that use a population of points in the effort to find the optimal solution. The stronger members of the population survive, mate and produce offspring that may undergo a mutation. These offspring form a new generation. Genetic algorithms have been used on sequencing problems before, although they cannot use natural crossover techniques in searching the solution space. The advantage of the problem space is that the genetic algorithm can use standard techniques to create offspring.

Our genetic algorithm searches the space of adjusted deadlines. We use a binary coding for the adjusted deadlines and a single pass of the Minimum Waste algorithm as the heuristic. For this genetic algorithm, we use many of the ideas presented in Davis (1991), to which we refer readers who wish to learn more about the issues discussed here.

In the problem space, each point is a vector of integers that are deadlines used as input for one pass of the Minimum Waste algorithm. We will use a binary representation of the points in problem space. In the population of the genetic algorithm, each individual is a string of bits. Each successive six-bit substring represents a deadline for a specific job. The integer decoded from this binary number ranges from zero to 63 and linearly maps to a real number in the range from zero to the maximum deadline in the given problem data. This discretization reduces the problem space but still allows the deadlines to vary significantly with respect to each other.

The adjusted deadlines are used as input to a single pass of the Minimum Waste algorithm, which outputs a sequence of jobs. The algorithm uses the largest of the adjusted deadlines as the initial makespan and schedules the jobs accordingly, using the actual job processing and class setup times where necessary but using the adjusted deadlines to determine when a job is eligible. If necessary, the algorithm can start jobs before time zero.

Using the actual problem data and the sequence of jobs output from one application of the Minimum Waste Algorithm, we can create a schedule of jobs that starts at time zero and has no

inserted idle time. We evaluate the original string of bits by computing the feasibility and the total flowtime of this schedule.

Since we cannot guarantee that the algorithm will produce a schedule with no tardy jobs, we use a penalty function to make undesirable those individuals in the population that yielded infeasible schedules (with respect to the actual deadlines). This penalty function is $\Phi = \sum T_j^2$, where $T_j = \max\ \{0, C_j - D_j\}$. Our objective function f is defined as $f = \sum C_j + r\ \Phi$.

In order to encourage solutions with good total flowtime (regardless of feasibility) at the beginning of the search and to encourage feasibility near the end of the search, we start the search with the constant r small and increase it periodically.

The initial population includes one individual (the dummy, or seed) that we create by dividing each actual deadline by the maximum deadline, multiplying by 63, rounding down to the nearest integer, and converting this integer (which is in the range 0 to 63) to its binary representation. The remaining individuals in the initial population are constructed by mutating the bits in the initial (dummy) chromosome. The mutation rate is set at fifty percent (0.5). Note that using an initial mutation rate of 0.5 is equivalent to choosing a random chromosome from the entire search space.

Let us illustrate this procedure using Example 3.1, the problem we introduced earlier (see Table 3.1 for problem data). Also, let us define $\lfloor x \rfloor$ as the greatest integer less than or equal to x.

Mapping the deadlines to the bit strings yields the dummy, shown in Table 3.4. Let us create another member on the initial population. If two bits, the fourth of the fourth substring and the first of the last substring, are flipped in the mutation, we have the point in problem space shown in Table 3.5. Performing one pass of the Minimum Waste algorithm on the new deadlines (see Table 3.6) yields the sequence [1 5 4 3 2], from which we create the feasible schedule shown in Figure 3.5, with a makespan of 15 and a total flowtime of 46.

Table 3.4. Bit representation of the dummy.

$\lfloor 63\, D_j/18 \rfloor$	10	56	49	35	63
Bits	001010	111000	110001	100011	111111

Table 3.5. A point in the problem space.

Bits	001010	111000	110001	100111	011111
Integer	10	56	49	39	31
D_j	2.85	16	14	11.14	8.85

Table 3.6. Application of the Minimum Waste algorithm.

	Waste:					Schedule:	
Time:	J_1	J_2	J_3	J_4	J_5	J_j	C_j
16	13.15	0	2	4.86	7.15	J_2	16
14	11.15		0	2.86	5.15	J_3	14
12	9.15			2	3.15	J_4	10
7	4.15				0	J_5	7
5	2.15					J_1	2.85

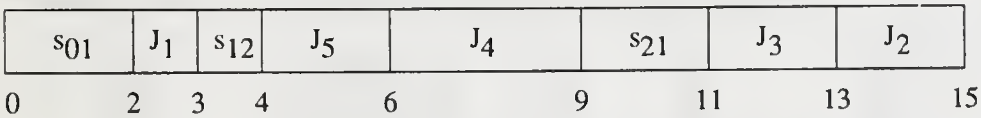


Figure 3.5. Schedule corresponding to new bit string.

After some experimentation we decided to use a steady-state genetic algorithm that created offspring by repeatedly (and randomly) selecting from a set of four genetic operators: one-point crossover, uniform crossover, and two types of mutation. *Small mutation* used a low probability (two percent per bit) of flipping a bit in the string; *large mutation* used a higher probability (fifty percent). The search used tournament selection for selecting the parents necessary for the crossover or mutation, and duplicate bit strings were not allowed in the population. Tuning was performed in order to determine some good settings for various algorithm parameters. (See Table 3.7. The problem sets used for tuning are described in Section 3.2.6.)

Table 3.7. Flowtime performance while tuning 2000-individual genetic algorithm on 30-job problems (Newprob4).

Population Size	Operator Probabilities (in percent)	Increase in *r*	Frequency of Increase	Mutation Rate	Average Ratio
Tuning Increase in *r* and Frequency of Increase, Population Size = 10					
10	25, 25, 25, 25	+2	10	2%	0.8575
10	25, 25, 25, 25	+10	10	2%	0.8666
10	25, 25, 25, 25	+50	10	2%	0.8562
10	25, 25, 25, 25	+2	50	2%	0.8646
10	25, 25, 25, 25	+10	50	2%	0.8700
10	25, 25, 25, 25	+50	50	2%	0.8624
10	25, 25, 25, 25	+2	100	2%	0.8676
10	25, 25, 25, 25	+10	100	2%	0.8755
10	25, 25, 25, 25	+50	100	2%	0.8625
Tuning Increase in *r* and Frequency of Increase, Population Size 50					
50	25, 25, 25, 25	+2	10	2%	0.8597
50	25, 25, 25, 25	+10	10	2%	0.8660
50	25, 25, 25, 25	+50	10	2%	0.8623
50	25, 25, 25, 25	+2	50	2%	0.8669
50	25, 25, 25, 25	+10	50	2%	0.8687
50	25, 25, 25, 25	+50	50	2%	0.8654
50	25, 25, 25, 25	+2	100	2%	0.8619
50	25, 25, 25, 25	+10	100	2%	0.8626
50	25, 25, 25, 25	+50	100	2%	0.8669
Tuning Increase in *r* and Frequency of Increase, Population Size 100					
100	25, 25, 25, 25	+2	10	2%	0.8800
100	25, 25, 25, 25	+50	10	2%	0.8794
100	25, 25, 25, 25	+2	50	2%	0.8738
100	25, 25, 25, 25	+50	50	2%	0.8792
100	25, 25, 25, 25	+2	100	2%	0.8751
100	25, 25, 25, 25	+50	100	2%	0.8716
Tuning Mutation Rate					
10	25, 25, 25, 25	+50	10	2%	0.8562
10	25, 25, 25, 25	+50	10	10%	0.8990
10	25, 25, 25, 25	+50	10	50%	0.9962
10	25, 25, 25, 25	+50	50	2%	0.8624
10	25, 25, 25, 25	+50	50	10%	0.8953
10	25, 25, 25, 25	+50	50	50%	0.9629
50	25, 25, 25, 25	+50	10	2%	0.8623
50	25, 25, 25, 25	+50	10	10%	0.8994
50	25, 25, 25, 25	+50	10	50%	0.9236

Table 3.7. (Continued)

Population Size	Operator Fitnesses (in percent)	Increase in *r*	Frequency of Increase	Mutation Rate	Average Ratio
Tuning Operator Fitness					
10	75, 0, 0, 25	+50	10	2%	0.8664
10	50, 15, 10, 25	+50	10	2%	0.8764
10	25, 25, 25, 25	+50	10	2%	0.8562
10	15, 50, 10, 25	+50	10	2%	0.8758
10	0, 75, 0, 25	+50	10	2%	0.8592
10	75, 0, 0, 25	+50	50	2%	0.8730
10	50, 15, 10, 25	+50	50	2%	0.8628
10	25, 25, 25, 25	+50	50	2%	0.8624
10	15, 50, 10, 25	+50	50	2%	0.8649
10	0, 75, 0, 25	+50	50	2%	0.8688
50	75, 0, 0, 25	+50	10	2%	0.8574
50	50, 15, 10, 25	+50	10	2%	0.8732
50	25, 25, 25, 25	+50	10	2%	0.8623
50	15, 50, 10, 25	+50	10	2%	0.8689
50	0, 75, 0, 25	+50	10	2%	0.8720
Tuning 3000-individual genetic algorithm on 50-job problems (Prob50z).					
10	25, 25, 25, 25	+50	50	2%	0.8880
50	25, 25, 25, 25	+50	50	2%	0.8931
50	25, 25, 0, 50	+50	50	2%	0.8739

Notes: Performance is average ratio to solution found by the Multiple-Pass Minimum Waste Heuristic. Operators are one-point crossover, uniform crossover, large mutation, and small mutation. Mutation rate is probability per bit.

These parameters included the population size, the mutation rate, the operator selection probabilities, and the rate of change of the penalty coefficient *r*. The rate of increase in the penalty coefficient affected the solution quality slightly. The mutation rate, population size, and relative probabilities of operator selection affected solution quality more significantly, with a smaller mutation rate (two percent, as mentioned earlier), smaller population sizes, and a higher probability for selecting the small mutation yielding better solutions. These factors imply that the search can easily find good neighborhoods (especially since a point corresponding to the original problem data is included in the initial population) but needs to spend time hunting for a better

solution. Thus, a search that incorporates some kind of local search at the end of the genetic algorithm may be useful.

3.2.6 Empirical Testing

In this section we describe a set of experiments performed in order to test the genetic algorithm and the heuristics. We discuss the generation of sample problems and the computational results.

Problem generation. In order to test the heuristics and the genetic algorithm described above, it is necessary to create a set of test problems. We describe in this section how we can create problems that have at least one feasible solution and problems where finding a feasible solution is more difficult.

The problems in the first problem set have 30 jobs in four classes, with random processing times in the range [1, 20] and sequence-dependent setup times in the range [0, 5]. For this set, we want to determine random deadlines in order to insure that some feasible schedule did exist.

We use the following procedure: after computing the random class setup times, each job is given a random processing time, and an initial completion time is computed by scheduling it after all previously constructed jobs. This first-generated, first-served schedule yields a makespan that becomes an upper bound for the deadlines, and each job is given a deadline determined by sampling a random variable uniformly distributed between the job completion time (in this schedule) and the makespan, i.e. the interval $[\, C_j, C_{max} \,]$. Thus, the initial sequence is a feasible solution.

In order to determine the performance of the genetic algorithm on minimizing the flowtime when feasible solutions are harder to locate, a number of additional problem sets are created. In addition to the problem set described earlier, which includes problems that were known to have a feasible solution, we generate 30-job and 50-job problems with tighter deadlines. The 30-job problems have four job classes and the 50-job problems ten job classes. Tighter deadlines are achieved by extending the range of values that a random deadline could take. Let us define a value k that can range from zero to one. The deadline for J_j is taken from the interval

$[k\,C_j, C_{max}]$, where the C_j and C_{max} are from the original generated schedule. If $k = 1$, this plan is the same as the original one, and the generated problem is guaranteed to have a feasible schedule. If $k = 0$, all of the deadlines vary equally, and there may exist no feasible schedule. As k decreases from one to zero, the problems we generate have a higher probability of having fewer feasible schedules. We generated problems with $k = 0, 0.2$, and 1. Since the Multiple-Pass Minimum Waste Heuristic cannot find feasible solutions for some of these problems, we will see if the genetic algorithm can find solutions that are feasible.

Results. In this section we discuss the results of our experiments with the solution procedures on the generated problem sets. We summarize the findings and present tables of the collected data. Since the optimal solutions are not known (a branch-and-bound algorithm to find optima requires excessive computational time) and no good lower bound can be determined, we measure the performance of the solution procedures relative to each other.

Each procedure was run once on each of the problems in the problem sets. The procedures include the Multiple-Pass Minimum Waste Heuristic and the problem space genetic algorithm.

For comparison purposes, we also implemented a version of the heuristic that Ahn and Hyun (1990) use to reduce the total flowtime in class scheduling problems. They proposed an iterative heuristic that starts with an initial feasible sequence where the jobs in each class are in SPT order (since they are not concerned with deadlines) and applies both a forward and backward procedure to it, repeating the steps until no strict improvement is found. Each of the forward and backward procedures interchanges different subschedules where the second subschedule consists of jobs from one class and the first subschedule has no jobs from this class. If the interchange reduces the total flowtime, the subschedules are switched; this maintains the class SPT property.

Our version of this algorithm, called the Modified Ahn & Hyun heuristic, uses one pass of the Minimum Waste algorithm to form the initial schedule. In addition, a potential swap of two subschedules is performed only if the swap reduces the total flowtime and maintains deadline feasibility.

The results (see Table 3.8) show that the genetic algorithm can find solutions that are much better than those found by the Multiple-Pass Minimum Waste Heuristic and are slightly better than those that Modified Ahn & Hyun heuristic produces. The genetic algorithm needs more time to find good solutions on the larger problems, although additional tuning may help improve the performance of the search.

The searches for the 30-job problems were for 2000 iterations using the following parameter settings: population size of 10, all operator fitnesses equal, increase of 50 in *r* every 10 individuals. The searches for the 50-job problems were for 3000 iterations using the same population size, no large mutations, and an increase of 50 in *r* every 50 individuals.

Table 3.8. Total flowtime performance of heuristics on problems where a feasible was found.

Problem Set	Problems	Jobs	*k*	Performance[a] of Heuristics Genetic Algorithm	 Modified Ahn & Hyun
Newprob4	10	30	1	0.8562	0.9130
Prob2a	4	30	0.2	0.9099	0.9346
Prob50z	10	50	1	0.8739	0.8755
Prob2c	8	50	0.2	0.8796	0.8914

Note: *a*: Performance is average ratio to solution found by the Multiple-Pass Minimum Waste Heuristic.

We observed that if the Multiple-Pass Minimum Waste Heuristic is unable to generate a feasible solution, the Modified Ahn & Hyun heuristic and the problem space genetic algorithm cannot locate a feasible solution. Thus our results are reported only for those problems where feasible schedules were created.

The genetic algorithm is able to outperform the Multiple-Pass Minimum Waste heuristic on total flowtime at the cost of increased computational time, as a full 2000-individual run lasts 256 seconds on average. (All computations performed on a 386 personal computer.) This is due to the effort of decoding the long bit strings into deadlines and the complexity of using one pass of the Minimum Waste algorithm to evaluate the individual. This computational effort is substantial

compared to that of running the Multiple-Pass Minimum Waste and Modified Ahn & Hyun heuristics. The heuristics need less than one second to find a good solution to a problem. A faster computer, however, would be able to reduce the computation time necessary for the genetic algorithm. Still, these results show that genetic algorithms that search the problem space can find very good solutions to scheduling problems.

The improvement in total flowtime of the solutions that the genetic algorithm can find is a result of two things: the multiple sampling of the search space and the evolutionary process. This leads to the following question: Are the genetic characteristics of the search a significant factor? We answer this question by changing the genetic algorithm so that each individual is a completely new one. Instead of choosing parents and creating offspring, we create new individuals by again mutating the dummy individual. This is a random sampling approach. (See Table 3.9.) These results imply that even though the same number of individuals are evaluated, the random sampling performs well but does not generate the same quality of solutions that the genetic algorithm does. Thus, we feel that the evolutionary process does contribute significantly to the improvement in total flowtime.

Table 3.9. Total flowtime performance of random sampling on problems where a feasible was found.

Problem Set	Problems	Jobs	*k*	Performance[a] of Random Sampling
Newprob4	10	30	1	0.9435
Prob2a	4	30	0.2	0.9545
Prob50z	10	50	1	0.9728
Prob2c	8	50	0.2	1.0165

Note: *a*: Performance is average ratio to solution found by the Multiple-Pass Minimum Waste Heuristic.

In conclusion, the genetic algorithm can find solutions with low total flowtimes. This good performance is due to the multiple sampling of the problem space (since more than one neighborhood can be searched at once); the use of the Minimum Waste algorithm to create

solutions from adjusted deadlines; and the ability of the genetic algorithm to combine the best characteristics of the points in the initial population.

3.2.7 Conclusions

This portion of the work has two contributions: it introduces an extended heuristic for the dual criteria class scheduling problem that we call CFTS, and it describes a problem space genetic algorithm used to find good solutions. The problem is to minimize the total flowtime subject to deadline constraints. In this section we present a multiple-pass heuristic for finding good solutions and discuss problem space and the genetic algorithm. Finally, we describe our experimental results, in which we compared the genetic algorithm to some heuristic approaches. From these results we make the following conclusions:

The Multiple-Pass Minimum Waste heuristic performs well at minimizing the total flowtime of CFTS. Though not an exact procedure, it is usually able to find feasible, high-quality solutions.

A genetic algorithm that searches a problem space of the Minimum Waste algorithm for CFTS can find solutions with lower total flowtime. This genetic algorithm includes a penalty function for infeasible points that increases the cost of tardiness as the search progresses. In addition, it produces slightly better solutions than another procedure modified for this problem.

3.3 Class Scheduling with Release and Due Dates

In this section, we study the one-machine class scheduling problem of minimizing the number of tardy jobs. Moreover, some of the jobs have non-zero release dates. We describe an extended heuristic developed for this problem and a genetic algorithm used to find good solutions. We also discuss an extension of this problem to the question of minimizing tardiness with minimum number of tardy jobs.

3.3.1 Introduction

The class scheduling problem studied in this section is to schedule a set of jobs, where some jobs have non-zero release dates, in order to minimize the number of tardy jobs. This problem is motivated by the semiconductor test area. Since post-assembly testing is the last stage in semiconductor manufacturing, meeting a job's due date is a very important objective for the manager of a test facility. The consideration of release dates is an attempt to model the look-behind situation that exists in the job shop, where the scheduling of a machine (the bottleneck, for instance) may be improved by including information about the jobs that are arriving soon.

This problem, like most class scheduling problems, is a difficult case. Since even finding a schedule with no tardy jobs is an NP-complete problem, exact algorithms to solve our problem in polynomial time do not exist. Thus, we are motivated to try different heuristics and searches.

Our approach was to modify an existing algorithm to include class setups and see how such an algorithm performs on this problem. Since the heuristic was not guaranteed to find good solutions, we also investigated a genetic algorithm. Thus, this research presents contributions in the extension of class scheduling problems to include a problem that has not been previously investigated and the use of both genetic algorithms and problem spaces to include the search for good solutions to class scheduling problems.

3.3.2 Literature Review

In this section we will mention some of the most relevant research on class scheduling and on the problem of minimizing the number of tardy jobs in the presence of release dates. A full discussion can be found in Chapter 2.

Bruno and Downey (1978) prove that, for general class scheduling problems, the problem of finding a schedule with no tardy jobs is NP-complete. Monma and Potts (1989) prove that class scheduling to minimize the number of tardy jobs is an NP-complete problem.

As discussed in Chapter 2, the one-machine problem of minimizing the number of tardy jobs when some have non-zero release dates ($1 / r_j / \Sigma U_j$) is a strongly NP-complete problem (Lawler, 1982). A restricted version of the problem has been considered by Kise, Ibaraki, and Mine (1978), who solve the problem optimally if the release and due dates match ($r_j < r_k$ implies $d_j \leq d_k$). They present an $O(n^2)$ algorithm (Kise's algorithm, described in Section 3.3.4) for this case.

3.3.3 Notation and Problem Formulation

We will use the basic notation introduced in Section 3.2.3. For CSRDD, each job J_j has a release date r_j and a due date d_j. For a given schedule, $U_j = 1$ if $C_j > d_j$ and 0 otherwise. The problem is to find a sequence that minimizes ΣU_j subject to the constraint that $C_j \geq r_j + p_j$.

An NP-complete problem, CSRDD is unstudied in the literature on class scheduling. In order to simplify the problem, it is assumed that the release and due dates match; that is, there exists an ordering where the jobs are simultaneously in Earliest Release Date (ERD) order and in Earliest Due Date (EDD) order. Our primary heuristic for CSRDD extends Kise's algorithm for the problem without setups to form a heuristic for finding good solutions.

3.3.4 Heuristics

In this section we will describe a number of heuristics: Kise's algorithm for the problem without class setups, our extension of this algorithm, and other heuristics used for testing purposes.

Kise's algorithm. Kise's algorithm orders the jobs by their release and due dates (a non-ambiguous ordering since the dates must match). The algorithm is an extension of the Moore-Hodgson algorithm (Moore, 1968) for minimizing the number of late jobs. Each job is scheduled after the partial schedule of on-time jobs while maintaining release date availability. If the new job is tardy, the algorithm searches the on-time jobs for the job whose removal leaves the shortest schedule of on-time jobs. The removed job is made tardy and will be processed with the other

tardy jobs after the feasible jobs. In this manner, the algorithm finds the largest subset of the jobs that can be delivered on-time. These jobs are scheduled in order of their release and due dates. The search subalgorithm has effort that is linear in the number of jobs in the partial schedule. Since the subalgorithm may be performed up to n times, the total effort of Kise's algorithm is $O(n^2)$.

Kise's heuristic is not optimal for CSRDD, although it can be modified to include setup times. Take the following example:

Example 3.3.

j	r_j	p_j	d_j	i
1	0	5	6	1
2	0	4	13	2
3	5	5	14	2
4	6	2	15	1

$s_{01} = s_{02} = 1$. $s_{12} = 1$. $s_{21} = 4$.

The optimal sequence is $[\, J_1\, J_4\, J_2\, J_3\,]$, with $C_1 = 6$, $C_4 = 8$, $C_2 = 13$, and $C_3 = 18$, which has one tardy job. Kise's algorithm adds J_3, which is tardy after J_1 and J_2, and the subalgorithm makes J_1 a tardy job. When J_4 is added to the schedule after J_2 and J_3, it also is tardy, for a total of two tardy jobs.

Kise extension. Our algorithm for CSRDD extends Kise's algorithm by considering two options when adding a new job to a partial schedule: we can place the job in a position after all of the on-time jobs or in a position after the last on-time job from the same class (if there is one). In either case, if an on-time job becomes tardy, we make tardy the job whose removal creates the shortest partial schedule of on-time jobs. We then choose between the two partial schedules created, selecting the new partial schedule with the smaller number of late jobs (or smaller makespan if they tie) as the incumbent before trying to schedule the next job.

Intuitively, it appears that the extended algorithm should outperform the Kise algorithm, since it includes an additional scheduling choice. Due to the complexity of the problem, however, this is not guaranteed. The following problem is one counter-example:

Example 3.4.

j	r_j	p_j	d_j	i
1	0	3	5	2
2	0	3	13	1
3	6	3	14	2
4	14	3	17	2

$s_{01} = s_{21} = 1$. $s_{02} = s_{12} = 2$.

The optimal sequence is [$J_1 J_2 J_3 J_4$], with $C_1 = 5$, $C_2 = 9$, $C_3 = 14$, and $C_4 = 17$, with none tardy. Kise's algorithm will construct this schedule. In the proposed algorithm, the addition of J_3 to [$J_1 J_2$] creates a partial schedule with no tardy jobs and a makespan of 14. The scheduling of J_3 after J_1 and before J_2 creates a makespan of 13 ($C_1 = 5$, $C_3 = 9$, $C_2 = 13$), so the sequence [$J_1 J_3 J_2$] replaces [$J_1 J_2 J_3$]. When we add J_4 after J_2, J_4 is tardy ($C_4 = 18$), and the on-time jobs complete at time 13; when J_4 is scheduled before J_2, J_2 is tardy ($C_4 = 17$, $C_2 = 21$). The algorithm thus yields [$J_1 J_3 J_2 J_4$], which is not an optimal schedule.

Tardiness rules. We also tested two heuristics based upon the R & M procedure of Rachamadugu and Morton (1982) and used for reducing weighted tardiness in Morton and Ramnath (1992). These are primarily class scheduling extensions.

RM: A dispatching rule where the priority of a job at time t is based upon the weight, the processing time, and the slack of the job:

$$\text{RM} = w_j / p_j * \exp(- S_j^+ / k * p_{avg}),$$

where RM is the job priority, w_j the job weight, p_j the processing time, S_j^+ the slack max $\{0, d_j - t - p_j\}$, k a predefined constant, and p_{avg} the average processing time of the jobs in the queue. In this formulation, jobs with higher weights, shorter processing times, and less slack will be scheduled first. According to Ramnath and Morton, the constant k is normally set to 2.

X-RM: The x-dispatch (look-behind) version of the RM rule includes jobs that will be arriving soon in an extended queue. That is, they arrive before the completion time of the shortest job already waiting. The RM priority is discounted by an amount that depends upon the arrival time:

$$\text{X-RM} = \text{RM} * (1 - (1.3 + p)*(r_j - t)^+/p_{min}),$$

where X-RM is the job priority, p the utilization factor, r_j the arrival time of the job, and p_{min} the smallest processing time among jobs currently available. Morton and Ramnath (1992) claim that this procedure reduces weighted tardiness by 40% over the standard RM rule.

These two priorities are used as dynamic dispatching rules. At a time t, the job with the highest priority is scheduled next. For our class scheduling problem, we redefine the components to include the setup times, but otherwise we use the same formulas. This works well for the objective of weighted tardiness, but for our objective (minimizing the number of tardy jobs), we would like to postpone the processing of the tardy jobs in order to concentrate on the on-time jobs. Thus, when we schedule a job that will be tardy, we look for the job whose removal will result in a shorter schedule with no tardy job and we remove that.

For our class scheduling problem, we include in the processing time the class setup necessary to process a job and modify the release date by the same amount. That is, if the job completing at time t is in G_i and J_j is in G_k, we add s_{ik} to p_j and subtract s_{ik} from r_j (for the X-RM calculation). We use $p = 1$ and $k = 2$ and $w_j = 1$ for all J_j.

3.3.5 Analysis of the Heuristic

In this section we discuss the computational effort necessary to perform the extended Kise heuristic and the worst case error of this heuristic. A pseudo-code presentation of the heuristic and its subalgorithm can be found in the Appendix.

It is obvious that the extended version of Kise's heuristic takes more effort than Kise's algorithm. However, the effort of the algorithm is still O(n^2). When adding the next job to a partial schedule, there are two positions for the new job. The algorithm would take at most O(n) effort to find the last job from the same class as the new job, to insert the new job, and to determine which (if any) jobs are now tardy. If there is a tardy job, a pass of Kise's subalgorithm must be performed to determine which job to remove. This is also O(n). For the other position, there is O(n) effort in adding the new job to the end of the partial schedule and performing a pass

of the subalgorithm. Thus, the total effort of adding the new job is O(n), and since n jobs must be scheduled, the total effort of the extended Kise heuristic is O(n^2).

Since CSRDD is strongly NP-complete, there is no optimal polynomial or pseudo-polynomial algorithm. Since our extended Kise heuristic is not guaranteed to find an optimal solution, we need to look into the worst-case error bound. In the following we describe two families of instances for CSRDD where the extended Kise heuristic cannot find good solutions. While the first of these examples prove that the heuristic can perform arbitrarily badly, the examples will also provide problem instances that we can use for testing the performance of the genetic algorithm. They are especially good for this since we know the optimal solution in advance.

Example 3.5. In this case, the extended Kise heuristic finds n - 1 out of n jobs tardy when the optimal has only two tardy jobs. There are n jobs where J_1 is in G_1 and $J_2, \ldots, J_n$ are in G_2, and the jobs have the following characteristics:

J_j	J_1	$J_2, \ldots, J_n$
r_j	0	1
p_j	1	1
d_j	1	n

The class setups are as follows:

G_i	G_1	G_2
s_{0i}	0	2
s_{1i}	-	n
s_{2i}	0	-.

The optimal sequence (for $n > 2$) begins with the n - 1 jobs in G_2. The last job ends at 1 + (n - 1) = n, so they are all on-time. Since J_1 is scheduled after this, it is tardy. The Kise and Kise-extension algorithms start by scheduling J_1 first. This job completes at time 1. When the first job from G_2 is scheduled to form [J_1 J_2], the job completes at 1 + n + 1 = n + 2 and is therefore tardy. Removing J_1 yields a partial schedule that ends at time 2 and is thus longer than the partial

schedule consisting of just J_1. Thus J_2 is made tardy. This continues for all of the jobs from G_2, and they are all forced to be tardy, for n - 1 tardy jobs.

Example 3.6. In this case the optimal solution has no tardy jobs and the extended Kise heuristic finds $n/3$ tardy jobs. We construct the problem instance in the following way: choose a non-negative integer k. Let $n = 3(k + 1)$. Let $m = 3$. For $i = 0, \dots, k$, construct three jobs, J_{3i+1} in G_1, J_{3i+2} in G_2, and J_{3i+3} in G_3, with the following job characteristics, where $0 < \varepsilon < 1$ and $0 < \delta < 1$:

J_j	J_{3i+1}	J_{3i+2}	J_{3i+3}
r_j	3i	3i	3i
p_j	$1-\varepsilon$	$1+\varepsilon$	1
d_j	3i+2	$3i+2+\delta$	3i+3.

Let the class setups be as follows, where $s > \delta$:

G_i	G_1	G_2	G_3
s_{0i}	0	0	0
s_{1i}	-	s	0
s_{2i}	0	-	s
s_{3i}	1	0	-.

The optimal sequence of jobs is [$J_2 J_1 J_3 J_5 J_4 J_6 \dots J_{3k+2} J_{3k+1} J_{3k+3}$]. This schedule has no inserted idle time and no setup time (see Figure 3.6).

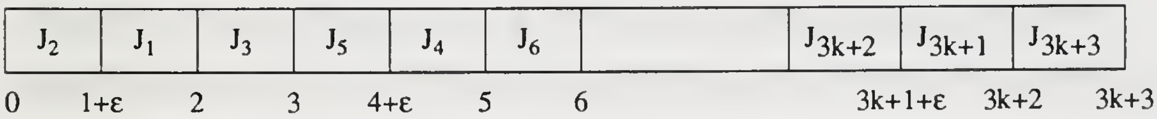


Figure 3.6. Optimal Schedule.

Taking the jobs in ERD order, we schedule J_1 first ($C_1 = 1 - \varepsilon$). J_2 is scheduled next, but $C_2 = C_1 + s_{12} + p_2 = 2 + s > d_2 = 2 + \delta$. Thus, J_2 is tardy, and since $p_2 > p_1$, J_1 remains while J_2 is now tardy. J_3 is added next, with $C_3 = C_1 + s_{13} + p_3 = 2 - \varepsilon$. J_4 is added next, starting at time $3 > C_3 + s_{31} = 3 - \varepsilon$ and completing at time $C_4 = 4 - \varepsilon$. (Starting J_4 after J_1 makes J_3 tardy.) By repeating the above argument, it can be shown that J_5 will be made tardy and J_6 will follow J_4.

This process continues for all of the jobs (see Figure 3.7). Thus, the heuristic creates a schedule where the $k + 1 = n/3$ jobs in G_2 are tardy, while the optimal schedule has no tardy jobs. Note that we never insert a job into the middle of a partial schedule; thus, Kise's algorithm and the extended Kise heuristic create the same schedule.

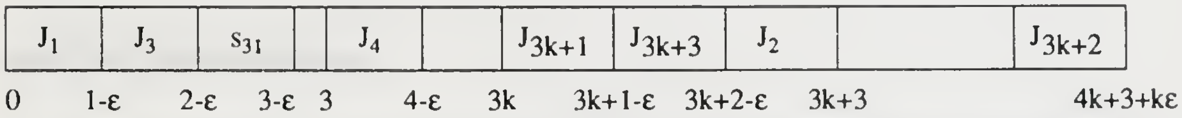


Figure 3.7. Heuristic Schedule.

The first of the two above cases will be useful in the proof of the following error bound theorem. For the purposes of Theorem 3.2, we assume that the triangle equality holds for the class setups: $s_{ab} + s_{bc} \geq s_{ac}$, for all $a = 0, \ldots, m$, $b = 1, \ldots, m$, $c = 1, \ldots, m$.

Theorem 3.2. For an instance of CSRDD, if there exists a schedule in which at least one job completes on-time, the extended Kise heuristic will create a schedule with at least one on-time job. Moreover, there exist problem instances where the heuristic will schedule exactly one on-time job, although there exist schedules with more than one on-time job.

Proof. First, we note that the number of scheduled on-time jobs never decreases as the extended Kise heuristic schedules new jobs. Now, consider a job J_j in class G_i that completes on-time in some feasible schedule. Thus, for J_j, $C_j \leq d_j$. Now, $C_j \geq r_j + p_j$ and $C_j \geq s_{0i} + p_j$, since by the triangle inequality stated above, if there exist any jobs before J_j, the sum of the setups before J_j is at least s_{0i}. Thus, $d_j \geq r_j + p_j$, and $d_j \geq s_{0i} + p_j$.

Suppose that the schedule created by the extended Kise heuristic has no on-time jobs. Then, when the extended Kise heuristic considered J_j, no other on-time jobs were scheduled. The heuristic started J_j as soon as possible (the maximum of r_j and s_{0i}), but J_j was tardy. Thus, $C_j = \max\{r_j, s_{0i}\} + p_j > d_j$, but this contradicts the result above. Thus, the heuristic creates a schedule with at least one on-time job.

Thus, we have shown that the extended Kise heuristic will schedule at least one on-time job. This bound is tight, as our discussion of Example 3.5 shows that there exist problems for which the heuristic will schedule exactly one on-time job although the optimal schedule has more than one on-time job.

3.3.6 The Genetic Algorithm

In this section we present the problem space and discuss the details of the genetic algorithm we used to find good solutions for CSRDD.

Problem space. In Chapter 2 we described the ideas of alternative search spaces. In this section we present the problem space that we searched in order to find good solutions for CSRDD.

We defined a problem space for CSRDD in the following manner: Given a problem p in problem space, a heuristic h is a function that creates a sequence corresponding to a solution s for CSRDD, i.e. $h(p) = s$. We defined a problem as a vector of job release dates, using a pass of Kise's algorithm to create a sequence of jobs by considering the jobs in order of their new release dates instead of the order imposed by the matching release and due dates. The actual release dates are used in determining the schedule, however. Note that all solutions for CSRDD (including the optimal ones) that schedule all tardy jobs last are in the range of h. Following are two examples of applying this heuristic to different vectors of deadlines for the problem in Example 3.3:

Heuristic(problem) = solution:

Kise (0, 1, 5, 6) = [$J_2 J_3 J_1 J_4$], two jobs tardy.
Kise (0, 8, 5, 4) = [$J_1 J_4 J_3 J_2$], one job tardy.

A genetic algorithm for CSRDD. In this work we developed a genetic algorithm based on the ideas presented in Davis, 1991, namely, steady-state reproduction without duplicates,

fitnesses measured by linear normalization, a uniform crossover operation, operation selection, and interpolated parameters.

Steady-state reproduction adds new individuals a few at a time, whereas the traditional method replaces the entire population with a new generation. Steady-state reproduction (attributed to Whitley, 1988, and Syswerda, 1989) is used to ensure that good individuals (and their good characteristics) survive. Steady-state reproduction without duplicates prevents children that are identical (in chromosome values) to a current member from joining the population.

Linear normalization is a fitness technique that creates fitness values by ordering the individuals in a population by their objective function evaluation. The assignment of fitnesses begins with a constant value and decreases the fitness linearly as it considers each individual in order. This technique prevents a super individual from dominating the population at the beginning of a run and yet differentiates between the various very good individuals that exist near the end of a run. Of course the values of the original constant and the decrement parameter influence the extent of these two phenomena.

Uniform crossover, an operator first described by Syswerda (1989), is a way to combine characteristics in ways that standard one- or two-point crossovers cannot. In a uniform crossover, two parents are selected and two children produced. Each bit position is considered independently and the parent that contributes the bit value for that position in the first child is determined randomly. The second child receives the value for that position from the other parent. While uniform crossover can destroy a good characteristic by mixing it with a bad string, it can also combine features that are widely dispersed across the string.

Since one-point crossover remains a good operator, however, we used both types of crossover in our genetic algorithm. Before creating a child, we randomly decide on which operator we wish to perform: uniform crossover, one-point crossover, or mutation. Each operator has an operator fitness and the probability of that operator being selected is proportional to that fitness. If one of the crossovers is selected, two parents are selected and two children are created.

If the mutation operator is selected, one parent is selected and a child created by forcing each bit to undergo a mutation with some small probability. The child or children created are checked against the current population for duplication, evaluated, and inserted into the population, replacing the worst members of the population. The new population is then reordered and new fitnesses created using the linear normalization technique.

As we did with the CFTS genetic algorithm, the initial population included one individual (the dummy, or seed) that was created form the actual problem data. The remaining individuals in the initial population were constructed by mutating the bits in the initial (dummy) chromosome. This initial mutation rate was set at 0.05 per bit.

We interpolate the following parameters over the course of the run: the decrement for linear normalization is increased, and the operator fitnesses are changed to favor crossovers and discourage mutations.

Table 3.10. Parameter values for genetic algorithm.

Population size: 100
Linear normalization decrement: 0.2 to 1.2
Probability of selecting mutation operator: 35% to 25%
Probability of selecting uniform crossover operator: 40% to 30%
Probability of selecting one-point crossover operator: 25% to 45%
Mutation rate: 0.02 per bit

3.3.7 Empirical Tests and Results

In this section we describe the empirical tests conducted to test how well the heuristics and genetic algorithm perform.

Problem generation. We created problem sets using the due date assignment method of Hariri and Potts (1989). The release dates can also be created using a similar method. In this method, setup times and processing times of the jobs are determined first from random variables. Then, an estimate of the maximum completion time is made by simply summing the job

processing times. This makespan is used to define a specific ranges for the due dates and a range for the release dates. The due dates and release dates are sorted and the matching pairs are given to the jobs. By changing the parameters governing the definition of the ranges, problem sets with different characteristics can be created.

Different sets of 10 problems were created. The number of jobs per problem ranged from 15 to 100. The processing times ranged from 1 to 20 and the class setup times from 0 to 9. The jobs were randomly placed into a number of job classes, depending on problem size. The release dates and due dates were taken from a uniform distribution. The upper and lower bounds of this distribution were proportional to the sum of the job processing times. The proportions changed for each problem set. (See Table 3.11.)

Table 3.11. Data on problem sets.

Set	Problems	Jobs	Classes	Release date range	Due date range
KH301	10	30	4	0 - 0.4	0.4 - 0.6
KH302	10	30	4	0 - 0.4	0.6 - 1.0
KH151	10	15	4	0 - 0.4	0.4 - 0.6
KH501	10	50	5	0 - 0.4	0.4 - 0.6
KH303	10	30	4	0 - 0.4	0.2 - 1.0
KH304	10	30	4	0 - 0.6	0.2 - 1.0
KHMIXED2	10	30	4	0 - 0.4	0.4 - 0.6
KHMIXED3	10	30	4	0 - 1.0	0.4 - 1.2
KHMIXED4	10	30	8	0 - 1.0	0.4 - 1.2

Results. After numerical testing on these forty problems, it appears that the Kise and Kise extension heuristics and the R & M heuristics are fairly equal. A number of other heuristics were unable to find as many on-time jobs. Note that the lower bound was derived by using Kise's algorithm while ignoring all setup times.

Table 3.12. Average performance of heuristics.

Set	Jobs	Lower Bound	Kise	Extended Kise	RM	X-RM
KH301	30	8.3	12.0	10.7	11.6	11.6
KH302	30	1.1	4.9	2.7	4.2	4.3
KH151	15	4.8	6.8	6.4	6.6	6.7
KH501	50	12.6	19.5	16.0	17.9	17.9

Note: Performance is the average number of tardy jobs found by that heuristic on the ten problems in each problem set.

In addition to the problems where the release and due dates matched, we created a set of 30-job problems where no such correspondence existed. These ten problems had the same characteristics as the problems in the set KH301. On these problems the R & M heuristic performed slightly better than the extended Kise heuristic.

Table 3.13. Average performance of heuristics, non-matching release and due dates.

Set	Jobs	Kise	Extended Kise	RM	X-RM
KHMIXED2	30	7.7	5.3	4.5	4.7
KHMIXED3	30	11.1	10.2	8.9	8.7
KHMIXED4	30	10.0	9.2	7.2	7.0

Notes: Performance is the average number of tardy jobs found by that heuristic on the ten problems in each problem set.

Since the heuristics were finding good solutions, we decided to test the problem space genetic algorithm on 18- and 30-job problems that could not be solved well by the heuristics. These problems were instances of the problem described above in Example 3.6, where the extended Kise heuristic creates a schedule where a third of the jobs are tardy, although an optimal schedule has no tardy jobs. These results show that the problem space genetic algorithm is able to find good solutions. (In some of the following graphs we present our results with the number of on-time jobs, since the objective of the genetic algorithm was to maximize the number of on-time jobs.)

Table 3.14. Average performance of heuristics, hard problems.

Set	Jobs	Kise	Extended Kise	Genetic Algorithm[a] 1000	3000
Hard A	18	6.0	6.0	0.6	-
Hard B	30	10.0	10.0	4.5	1.5

Notes: Performance is the average number of TARDY jobs found by that heuristic on the five problems in each problem set.
a: Results reported at 1000 and 3000 individuals.

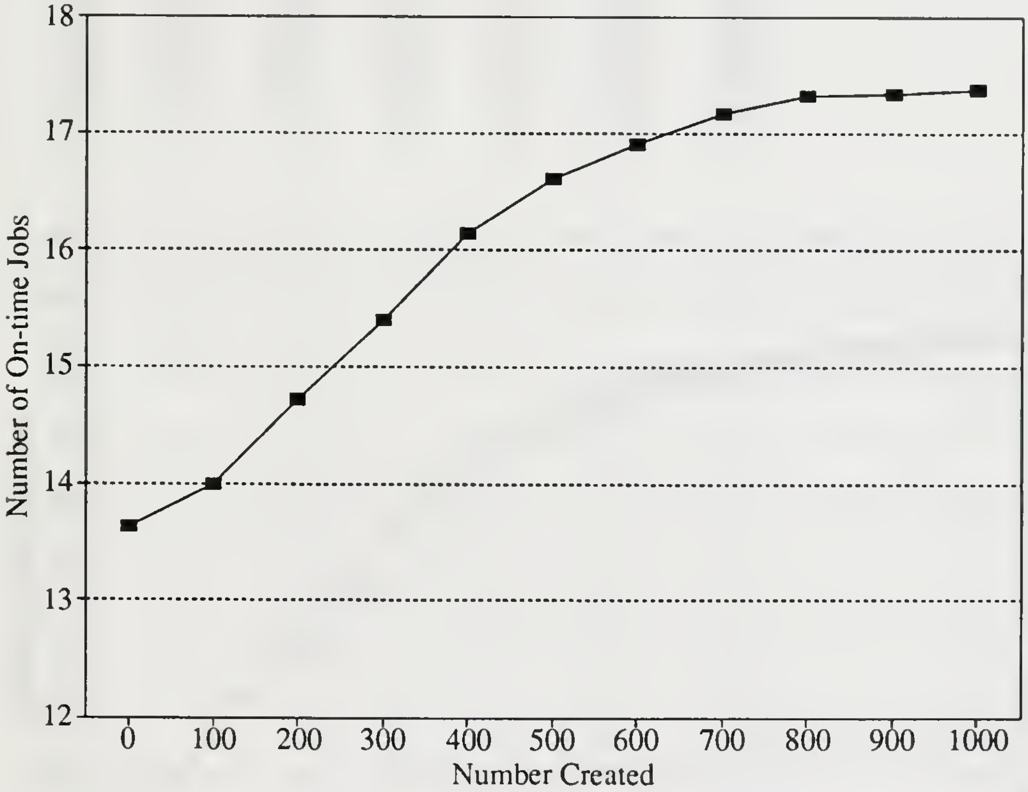


Figure 3.8. 18-job problems, Average number of ON-TIME jobs.

Table 3.15. 18-job problems, number of ON-TIME jobs, 10 runs of 1000:

Number of Individuals Created	Problem 1	2	3	4	5	Average
0	13.7	13.9	13.5	13.6	13.4	13.6
100	13.9	14.1	14.1	14.0	13.9	14.0
200	14.7	14.5	14.7	14.8	14.9	14.7
300	15.4	15.3	15.5	15.5	15.3	15.4
400	16.2	16.1	16.1	16.5	15.8	16.1
500	16.5	16.5	16.9	16.6	16.5	16.6
600	16.8	16.8	17.1	17.0	16.8	16.9
700	16.9	17.3	17.3	17.3	17.0	17.2
800	17.0	17.5	17.3	17.3	17.5	17.3
900	17.0	17.5	17.4	17.3	17.5	17.3
1000	17.1	17.5	17.5	17.3	17.5	17.4

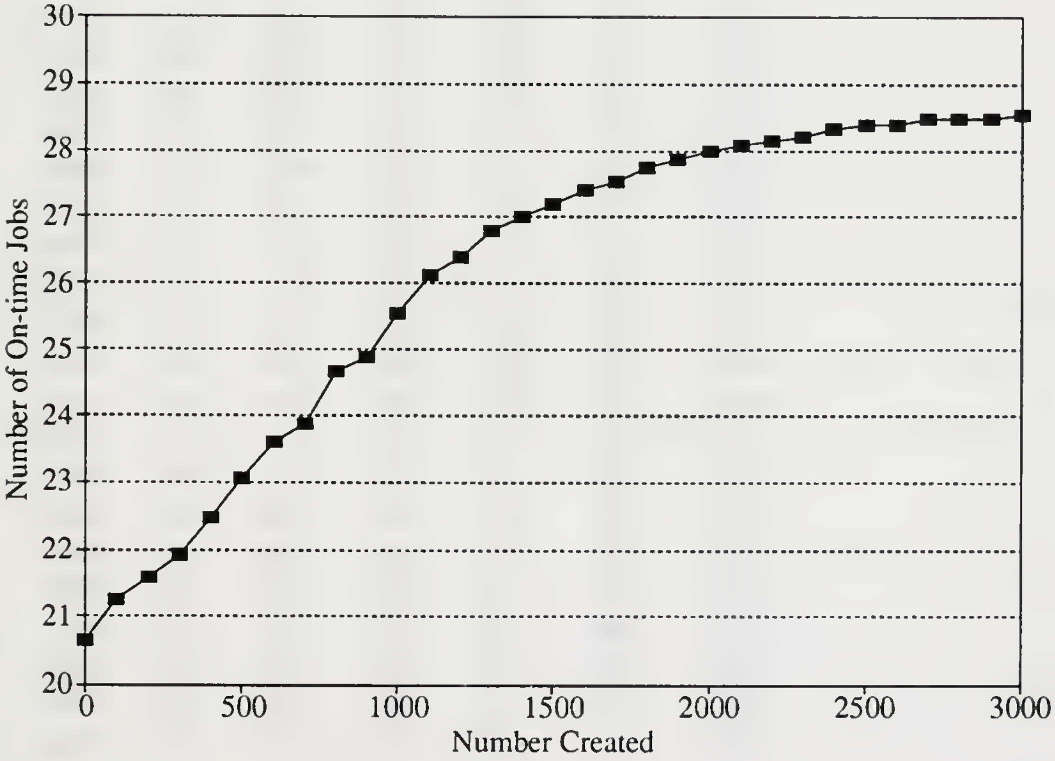


Figure 3.9. 30-job problems, average number of ON-TIME jobs.

Table 3.16. 30-job problems, number of ON-TIME jobs; 3 runs of 3000

Number of Individuals Created	Problem					Average
	1	2	3	4	5	
0	20.3	20.3	20.7	21.3	20.7	20.67
100	21.3	21.0	21.0	21.7	21.3	21.27
200	21.7	21.7	21.0	22.0	21.7	21.60
300	22.0	22.0	21.7	22.0	22.0	21.93
400	22.7	22.3	22.3	22.7	22.3	22.47
500	23.3	23.0	22.7	23.0	23.3	23.07
600	24.0	24.0	23.0	23.3	23.7	23.60
700	24.0	24.0	24.0	23.7	23.7	23.87
800	25.0	24.3	24.7	24.7	24.7	24.67
900	25.0	24.7	25.0	24.7	25.0	24.87
1000	25.7	25.0	26.3	25.3	25.3	25.53
1100	26.0	26.0	26.7	26.3	25.7	26.13
1200	26.3	26.0	27.0	26.7	26.0	26.40
1300	26.7	26.7	27.7	26.7	26.3	26.80
1400	27.0	26.7	28.0	26.7	26.7	27.00
1500	27.3	27.0	28.0	27.0	26.7	27.20
1600	27.3	27.3	28.0	27.3	27.0	27.40
1700	27.7	27.3	28.3	27.3	27.0	27.53
1800	28.0	27.7	28.3	27.3	27.3	27.73
1900	28.0	27.7	28.3	28.0	27.3	27.87
2000	28.0	27.7	28.3	28.3	27.7	28.00
2100	28.3	27.7	28.3	28.3	27.7	28.07
2200	28.3	27.7	28.3	28.3	28.0	28.13
2300	28.3	27.7	28.3	28.3	28.3	28.20
2400	28.3	28.3	28.3	28.3	28.3	28.33
2500	28.3	28.3	28.3	28.7	28.3	28.40
2600	28.3	28.3	28.3	28.7	28.3	28.40
2700	28.3	28.3	28.7	28.7	28.3	28.47
2800	28.3	28.3	28.7	28.7	28.3	28.47
2900	28.3	28.3	28.7	28.7	28.3	28.47
3000	28.3	28.7	28.7	28.7	28.3	28.53

3.3.8 Extension to Minimizing Tardiness

In addition to simply minimzing the number of tardy jobs, it is often an objective of schedulers to minimize the total tardiness of the tardy jobs. To this end, we study the problem of

minimizing the total tardiness subject to a constraint on the number of tardy jobs, since the minimization of total tardiness usually leads to schedules where many jobs are tardy and are tardy by a small amount. Since finding the minimum number of tardy jobs is an NP-complete problem, we use our heuristic to set the value of the constraint. Then, within that limitation, we minimize the total tardiness.

We develop some further extensions of Kise's algorithm that create a set of tardy jobs and then insert the tardy jobs into the schedule of on-time jobs in order to reduce the total tardiness. We also use our genetic algorithm to search for schedules with low number of tardy jobs and low total tardiness.

The problem of minimizing tardiness subject to a minimal number of tardy jobs has been considered for the problem without class setups (or release dates) by Vairaktarakis and Lee (1993), who develop an algorithm to optimally schedule a given set of tardy jobs and an efficient branch-and-bound technique to find the optimal tardy set. Other researchers have studied dual criteria problems with the same primary objective. Emmons (1975) considered the problem of minimizing total flowtime subject to minimum number of tardy jobs, using a branch-and-bound algorithm to find optimal solutions. Shanthikumar (1983) examined the problem of minimizing the maximum lateness subject to minimum number of tardy jobs, also using a branch-and-bound algorithm.

Our problem, which includes both class scheduling and non-zero release dates, is an NP-complete problem. The tardiness heuristics that we use to find good solutions use the extended Kise heuristic to determine a set of tardy jobs. The heuristics also use the sequence of on-time jobs created by the extended Kise heuristic, pushing the jobs to the right, starting them as late as possible, and attempting to insert the tardy jobs into the gaps in this schedule.

The first heuristic (T_1) orders the tardy jobs by their release dates and attempts to interleave the two sequences of jobs, scheduling tardy jobs to start as soon as possible while maintaining the feasibility of each on-time job (whose completion time is constrained by the due date).

The second heuristic (T_2) considers every tardy job as a candidate for a gap between the partial schedule and the next on-time job, selecting the tardy job that yields te earliest start time for the next on-time job. Any remaining tardy jobs are scheduled by their release dates.

The third heuristic (T_3) was a modification of the second that scheduled the remaining tardy jobs using a version of the Minimum Waste heuristic (see Section 3.2) that didn't consider deadlines (since all jobs are tardy). This heuristic has been shown to perform well on flowtime criteria. We use this because minimizing the total tardiness of the set of tardy jobs is identical to minimizing the total flowtime of those jobs.

Due to the non-optimal nature of the extended Kise heuristic, it is possible that the tardiness heuristics will often be able to schedule a tardy job so that it finishes on-time (reducing its tardiness to zero). However, the primary objective of these heuristics is to reduce tardiness, not reduce the number of tardy jobs.

Results. We tested the heuristics on three problem sets, selected because the variance of the due dates meant that the schedules were more likely to have gaps in which to insert tardy jobs. Each solution technique was measured by the average total tardiness found by that heuristic on the ten problems in each problem set and the percent deviation of this average from the average tardiness found by the extended Kise heuristic. The performance of the genetic algorithm on each problem is the average of ten trials of one thousand new individuals.

Table 3.17. Data on new problem sets.

Set	Problems	Jobs	Classes	Release date range	Due date range
KH303	10	30	4	0 - 0.4	0.2 - 1.0
KH304	10	30	4	0 - 0.6	0.2 - 1.0

Table 3.18. Average performance of heuristics.

Set	Extended Kise	T_1	%	T_2	%	T_3	%
KH302	236.4	237.3	-0.38	234.7	0.72	231.5	2.07
KH303	758.3	756.6	0.22	718.4	5.26	704.1	7.15
KH304	920.0	869.0	5.54	859.4	6.59	844.0	8.26

Note: Performance is the average total tardiness and the percent improvement.

Table 3.19. Average performance of heuristics.

Set	Extended Kise	G.A.	%	G.A. 3000	%
KH302	236.4	229.0	3.13		
KH303	758.3	884.3	-16.62	695.3	8.30
KH304	920.0	771.9	16.10		

Note: Performance is the average total tardiness and the percent deviation.

3.3.9 Conclusions

In this section we have introduced a class scheduling problem that we call CSRDD. The problem is to minimize the number of tardy jobs where some jobs have non-zero release dates, and we assume that the release and due dates match. We have described a heuristic developed to find good solutions. We have discussed a problem space and a genetic algorithm to search this space. We have described our experimental results, from which we make the following conclusions:

Our extended Kise heuristic can find good solutions for instances of CSRDD. It can do this by considering the class setups in both the subalgorithm that decides on which job to make tardy when a new job is added and the option to insert the new job into the middle of the partial schedule in order to reduce the number of setups.

When the extended Kise heuristic cannot find good solutions, our problem space genetic algorithm can. By searching the problem space near the original problem, it can discover solutions that are improvements on the schedule constructed by Kise's algorithm.

Also in this section we discussed an extension of this problem to the problem of minimizing total tardiness in the presence of a constraint on the number of tardy jobs.

3.4 Flowtime with Setups and Release Dates

The third of the class scheduling problems that we consider has jobs with non-zero release dates, and the objective is to minimize the total flowtime. We develop some lower bounds and dominance properties and examine some heuristics for finding good solutions to the problem. We discuss a problem space genetic algorithm that can improve the performance of a look-behind dispatching rule. For this problem we also developed a search technique for comparison purposes.

3.4.1 Introduction

This problem, like the others we have examined, is motivated by considering the scheduling of a semiconductor test area. We have class setups, arriving jobs, and an objective that mirrors the goal of management to minimize work-in-process inventory.

We will examine the problem of minimizing total flowtime when the jobs have non-zero release dates. This strongly NP-complete problem is a look-behind scheduling model, where we are interested in scheduling a machine by considering the jobs that will be arriving at the machine soon.

In addition to a look-behind scheduling rule, we will consider the use of a problem space genetic algorithm (similar to those developed for CFTS and CSRDD) that can improve the performance of this rule by adjusting the parameters of the rule. Due to the structure of the FTSRD problem, we will also use a decomposition heuristic as a means of comparing solution

quality. The decomposition heuristic is a search technique that considers a sequence of subproblems at each move.

In the next section we will introduce the notation and problem formulation. After that we will mention some of the previous research on the scheduling problem under consideration (a review of the literature on class scheduling and genetic algorithms can be found in Chapter 2), examine some lower bounds and dominance properties that can be used in a branch-and-bound technique, discuss our heuristics (including sequencing rules and the genetic algorithm), and report on the experimental results.

3.4.2 Notation and Problem Formulation

We use the same notation as that for the CFTS and CSRDD problems (Sections 3.2.3 and 3.3.3), except that the jobs do not have due dates. The FTSRD problem is to find a sequence that minimizes $\sum C_j$ subject to the constraint that $C_j \geq r_j + p_j$. We will see that FTSRD is NP-complete and has not been previously considered in the literature on class scheduling.

We make two assumptions in the analysis of the problem. One, a class setup for a job can begin before the job is available. Two, although all of the release dates are known, the processing for a job cannot begin until the release date. These conditions are motivated by our consideration of the one-machine problem as part of the job shop scheduling problem.

3.4.3 Background

The one-machine problem of minimizing total flowtime when the jobs have non-zero release dates has been previously studied in the case where no sequence-dependent setups are present. The problem is simple if the jobs are preemptive, that is, if a job that has begun processing can be interrupted by another job and then resumed later. In this case, the optimal policy at the next decision point is to schedule the job with the shortest remaining time. The set of decision points includes all job release times and completion times. If the jobs are non-

preemptive and have no sequence-dependent setup times, the problem (denoted by $1 / r_j / \Sigma C_j$) is a strongly NP-complete question, as shown by Lenstra et al. (1977).

Among class scheduling problems, minimizing the total flowtime is a strongly NP-complete problem (Monma and Potts, 1989). Thus, it can be seen that our problem, which is to minimize total flowtime with class setups and release dates, is also a strongly NP-complete task.

Many researchers (see Chapter 2) have considered the problem of minimizing the total flowtime subject to job release dates, $1 / r_j / \Sigma C_j$. Branch-and-bound algorithms have been the most popular approaches. None of these researchers included sequence-dependent setup times in their analysis.

3.4.4 Solution Techniques

In this section we will discuss a number of different approaches to solving the FTSRD problem. These include branch-and-bound searches, dispatching rules, a decomposition heuristic, and a genetic algorithm. A job *arrives* at r_j and is *available* at time t if $t \geq r_j$. Recall that we assume that a class setup can be performed before the associated job becomes available.

Branch-and-bound. A straightforward branch-and-bound algorithm can be developed for this scheduling problem. The branch-and-bound procedure finds the optimal solution by searching a tree that consists of every possible permutation. Each node of the tree consists of a partial schedule of jobs processed in that order and as soon as possible. (The root node consists of an empty schedule.) Unscheduled jobs will be appended to the schedule for a node. Branching occurs by adding a child node for each unscheduled job. A lower bound can be calculated for each node, and the search moves to the child with the lowest lower bound.

Local dominance properties. The number of nodes to be examined can be reduced by applying dominance properties that identify nodes that cannot lead to optimal solutions. Local dominance properties claim that a node α is dominated by another node β under certain conditions. Thus, we do not need to branch on node α because we can find a better solution by branching on β and searching its descendents.

We note here that matching processing times and release dates do not imply a dominance property. Even if $p_j < p_i$ implies $r_j \le r_i$ (or even if the release dates are identical) within each class, we do not have an optimal order for the jobs in each class. Consider the following instance:

	J_j	p_j	r_j
G_1	J_1	3	0
	J_2	4	0
	J_3	4	0
G_2	J_4	1	6
	J_5	1	6
	J_6	1	6

$$s_{01} = s_{21} = s_{02} = s_{12} = 1.$$

The optimal solution is $[J_2, J_4, J_5, J_6, J_1, J_3]$, with a total flowtime of 59. The best solution in which G_1 is in SPT order are $[J_1, J_4, J_5, J_6, J_2, J_3]$, with a total flowtime of 60 (see Figure 3.10).

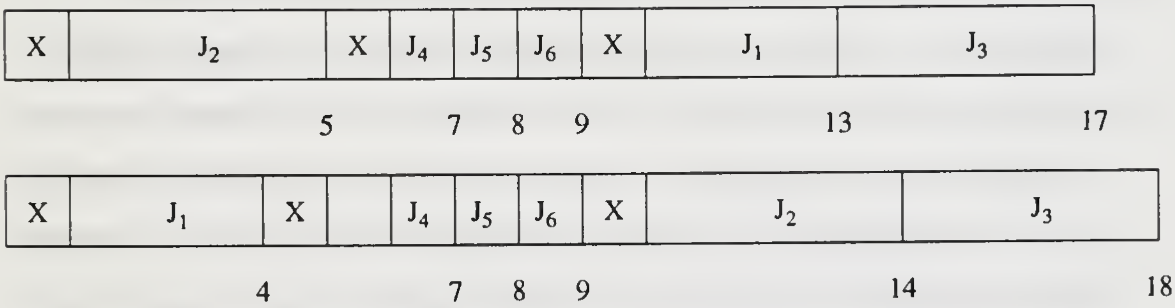


Figure 3.10. Optimal schedule and best SPT schedule.

A number of dominance properties have been suggested for the $1 / r_j / \Sigma C_j$. We have modified the properties of Dessouky and Deogun (1981) for use in our search. We make the assumption here that the class setups satisfy the triangle inequality: $s_{ac} \le s_{ab} + s_{bc}$ for all G_a, G_b, G_c.

Suppose that we are at a node in the search tree with a partial schedule σ that ends at time t with a job in class G_c and a set K of unscheduled jobs. If J_j is in K and in G_b, define the following *earliest start time*:

$$t_j = \max \{t + s_{cb}, r_j\}.$$

Note that if $t + s_{cb} \leq r_j$, the necessary class setup can be completed before the job arrives. At this node we can use any of the following dominance properties:

Property 3.1. Let J_i be the shortest job in K. If J_j is in G_b and J_i is in G_a, the node (σ, J_j) is dominated by (σ, J_i) if $t_i + s_{ab} \leq t_j$.

Property 3.2. The node (σ, J_j) is dominated by (σ, J_i) if $t_i + p_i + s_{ab} \leq r_j$.

Property 3.3. If J_j is in G_b and J_i is in G_a, the node (σ, J_j) is dominated by (σ, J_i) if all of the following statements are true:

(a) $t_i + p_i \leq t_j + p_j$,

(b) $t_i + p_i + s_{ae} \leq t_j + p_j + s_{be}$ for all job classes $G_e : G_e \cap K \neq \{\}$,

(c) $s_{eb} + p_j \leq s_{ea} + p_i$ for all $G_e : G_e \cap K \neq \{\}$, and

(d) $s_{eb} + p_j + s_{bd} \leq s_{ea} + p_i + s_{ad}$ for all $G_e : G_e \cap K \neq \{\}$, and $G_d : G_d \cap K \neq \{\}$.

Proofs: Property 3.1. Take any schedule $(\sigma, J_j, \sigma_1, J_i, \sigma_2)$, where there are m jobs in σ_1. If we move J_i before J_j, the new completion time of J_i is no greater than the old completion time of J_j; thus the flowtime of J_i is decreased by at least $(m + 1)p_i$ (since J_i is the shortest job). Meanwhile, we delay only the start of J_j and the m jobs in σ_1 by at most p_i (since $t_i + p_i + s_{ab} - t_j \leq p_i$). The jobs in σ_2 are not delayed at all (the triangle inequality of the setups insures this).

Property 3.2. Take any schedule $(\sigma, J_j, \sigma_1, J_i, \sigma_2)$. We can move J_i before J_j without delaying J_j, decreasing the flowtime of J_i and possibly that of the jobs in σ_2.

Property 3.3. Take any schedule $(\sigma, J_j, \sigma_1, J_i, \sigma_2)$. Note that the earliest start times of J_i and J_j are before σ_1. Interchange J_i and J_j. The new completion time for J_i is not greater the old completion time of J_j (by condition a). By condition b (if the first job of σ_1 is in G_e), the jobs in σ_1 are not delayed. Then, by condition c (if the last job of σ_1 is in G_e), the new completion time

for J_j is not greater than the old completion time of J_i. Finally, the last condition (if the first job of σ_2 is in G_d) implies that no jobs in σ_2 are delayed.

In any of these cases, we can take a schedule that starts with (σ, J_j) and find a schedule which starts with (σ, J_i) and which has less total flowtime. Thus, it is clear that the first node is dominated and that we do not need to search that part of the tree.

Lower bound. A branch-and-bound procedure needs a lower bound. A good lower bound is important to efficiently finding solutions. We develop two bounds: the first concentrates on the release dates, the second on the setup times.

The first lower bound for a node completely ignores the class setups. Instead, it solves the associated $1 / r_j$, preemption / $\sum C_j$ problem with the SRPT rule and adds the optimal flowtime to that of the partial schedule in the node.

The second lower bound separates the setup times from the job processing times. We assume that each class will have exactly one remaining setup. If we further assume that this setup will the shortest possible, then we can easily sequence the job classes to minimize the contribution of the setup times to the total flowtime. The unscheduled jobs are scheduled by SPT without regard to their release dates. The two sums are added to the flowtime of the partial schedule in the node.

The first lower bound should work well when the interarrival times are large, and the second bound should be useful at nodes where all of the unscheduled jobs are available.

Testing. For 30-job problems, the branch-and-bound procedure required excessive computation time to find an optimal. Still, improved lower bounds could be found by truncating the branch-and-bound search in the following way: We prevent the search from moving below a certain depth in the tree and take the lower bound at this depth as the objective function value. If we continue to do this, the truncated search returns the lowest lower bound at this depth. This will be a lower bound on the optimal solution and will be better than the lower bound computed at the root node.

Dispatching rules. The SPT rule is known to minimize the total flowtime for $1 // \Sigma C_j$ (Smith, 1956). Since we are studying a total flowtime problem, we are most interested in SPT-like heuristics. We will develop a rule that considers the waste associated with a job (the *waste* is the sum of the idle time and setup time incurred if the job is scheduled next). We propose the look-behind dispatching rule Shortest Waste, "Among the jobs with the minimum waste, schedule the shortest one." (This rule is similar to the Minimum Waste algorithm for CFTS.)

Let us define a few relevant variables: t is the current time, the completion of the last scheduled job; G_c is the class of the last scheduled job, and the *waste* of an unscheduled job J_j in G_b is

$$w_j = \max \{r_j - t, s_{cb}\}.$$

Our dispatching rule can be now stated:

Shortest Waste:

Among all unscheduled J_j, select the job with the minimum w_j. Break any ties by selecting the one with the minimum p_j.

Decomposition. For the FTSRD problem, we decided to implement a search heuristic for comparison purposes. A decomposition heuristic for finding good solutions to sequencing problems was introduced by Chambers et al. (1992). The heuristic is a type of local search. It begins with an initial sequence, and forms new sequences until it finds a local minimum. The critical step is the decomposition of the problem into subproblems that depend upon the current solution. A new solution is generated by combining the optimal or near-optimal solutions to each subproblem. The heuristic thus makes very good moves through the search space; only a few moves are needed before convergence is reached.

Consider, for example, a 12-job problem. Start with some initial solution to the problem. Select the first six jobs of this solution and find a good solution to the 6-job subproblem. Take the first three jobs (in order) of this subproblem solution as the first three jobs of the new solution. Then combine the remaining three jobs from the subproblem and with the next three

jobs from the initial solution. This forms a new 6-job subproblem. Continue solving subproblems and building the new solution until all of the jobs have been considered.

We use this technique in order to find good solutions against which we could measure the performance of our other heuristics. (We did not feel that this type of search would be as effective on deadline-oriented CFTS and CSRDD problems.) We use a branch-and-bound technique with an approximate dominance property to generate near-optimal solutions to the subproblems. The algorithm has two parameters. We need to select m as the size of the subproblems which we will solve; the larger the value, the better our subproblem solutions will be (at the expense of computation time). We also select f as the number of jobs from the subproblem solution that will be fixed into the new solution. A smaller f requires that more subproblems be solved per step. The following steps outline the procedure.

```
Step 1. Set $m$ and $f$. (We want $f$ to divide $n - m$.) Let $\sigma$ be the ERD schedule.
Step 2. Take the first $m$ jobs of $\sigma$. Let $\pi$ be an empty schedule.
Step 3. Solve the $m$-job subproblem by branch-and-bound.
Step 4. Append the first $f$ jobs of the solution to $\pi$.
Step 5. If there are any unconsidered jobs in $\sigma$, take the next $f$ jobs from $\sigma$, add to the
        $m - f$ remaining from the subproblem, and go to Step 3; else go to Step 6.
Step 6. Append the $m - f$ remaining jobs of the solution to $\pi$. If $\pi$ is a better schedule
        than $\sigma$, let $\sigma = \pi$ and return to Step 2; else go to Step 7.
Step 7. The solution found by the heuristic is $\sigma$.
```

We experimented with different values of (m, f). The best results (in terms of computation time and solution quality) were achieved with (9, 3) and (15, 5). We used our branch-and-bound algorithm with only the second lower bound and only the following approximate dominance property:

Property 3.4. Given a partial schedule σ, (σ, J_i) dominates (σ, J_j) if J_i and J_j are in the same class, J_i is the shortest unscheduled job in that class, and $w_i \leq w_j$.

This property is a simple extension of a previously considered dominance rule (see, for instance, Dessouky and Deogun, 1981). Unlike the dominance rules that we use in the full

branch-and-bound procedure, it has the advantage of being quick to check, since there are fewer unscheduled jobs from the same class.

In the section on computational results, we will discuss how well the decomposition heuristic performs.

A problem space genetic algorithm. In this problem, we consider the problem space defined over the problem release dates. A point in this space is an *n*-element vector of non-negative real numbers. When a heuristic is applied to an instance of FTSRD, it uses the actual release dates to generate a solution. If, however, we adjust the release dates of the problem, we can change the sequence created by the heuristic. This sequence can be evaluated as a schedule by using the actual problem data. We can associate, therefore, with the vector of adjusted release dates a performance value: the total flowtime of the schedule that was created. Moreover, we can search the space of adjusted release dates to find good schedules. This exploration is the objective of the problem space genetic algorithm. Our purpose is to show that the performance of a simple heuristic can be improved with a smart-and-lucky search like a genetic algorithm.

We will use the Shortest Waste heuristic to convert a vector of adjusted release dates into a sequence of jobs. The optimal solution is within the range of this heuristic: if each adjusted release date equals the actual start time of the job in an optimal solution, the Shortest Waste heuristic will schedule the jobs in the optimal order, since at any time, the job with the shortest waste will be the one with the next adjusted release date, which is the job with the next optimal start time.

As we did for CFTS and CSRDD, we will use a steady-state genetic algorithm. The initial population is formed by mutating a source individual that is the digital representation of the actual release dates. After empirical testing on a number of problem instances, we decided on the following parameters: The population size is 100 individuals. The four operators are uniform crossover, one-point crossover, small mutation, and large mutation; all have the same probability of being selected. In the small mutation, a bit is flipped with 2% probability; in the large, the probability increases to 50%. The algorithm uses tournament selection to identify parents. See

Davis (1991) or Goldberg (1989) for more information about these aspects of the genetic algorithm.

Example 3.7. The following problem is used to illustrate some of the issues we have discussed so far.

j	r_j	p_j	i
1	0	5	1
2	0	4	2
3	5	5	2
4	6	2	1

$s_{01} = s_{02} = 1.$ $s_{12} = 1.$ $s_{21} = 4.$

The ERD sequence is $[J_1\, J_2\, J_3\, J_4]$, with a total flowtime of 55. The EFT sequence is $[J_2\, J_3\, J_4\, J_1]$, with a total flowtime of 52. The Shortest Waste sequence is identical in this case. If we adjust the release dates to (1, 2, 5, 6), the Shortest Waste sequence is $[J_1\, J_4\, J_2\, J_3]$, with a flowtime of 45.

In the branch-and-bound algorithm, we compute lower bounds at the root node. The first lower bound for the entire problem is the SRPT schedule, total flowtime of 39, shown below (J_1 is preempted at time 6 by J_4):

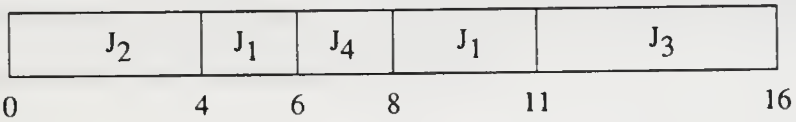


Figure 3.11. The SRPT schedule.

The second lower bound (Longest Weighted Batch Size plus SPT) is computed as follows:

Batch sizes: $b_1 = 2.$ $b_2 = 2.$
Shortest setups: $s_1 = s_{01} = 1.$ $s_2 = s_{02} = 1.$
$\sum s_i B_i = 1(4) + 1(2) = 6.$
Processing times in SPT order: 2, 4, 5, 5.
Completion times: 2, 6, 11, 16. $\sum C_j = 35.$
Lower bound = 35 + 6 = 41.

3.4.5 Empirical Testing

Problem generation. In order to test the heuristics, four sets of ten problems were created. The characteristics of the sets are shown below (processing, interarrival, and setup times randomly selected from uniform distributions with the given ranges):

Table 3.20. Data on problem sets

Set	Problems	Jobs	Classes	Processing Times	Interarrival Times	Setup Times
FT151	10	15	5	1,20	1,10	0,9
FT301	10	30	5	1,20	1,15	0,9
FT302	10	30	10	1,20	1,15	5,9
FT304	10	30	10	1,15	1,20	5,9

Results. In this section we will discuss how well our solution techniques performed. (See Table 3.21.) The branch-and-bound could find optimal solutions on only the 15-job problems. On the 30-job problems, we used the decomposition heuristic with parameters (9, 3) to quickly generate solutions and measured the performance of other heuristics against these solutions. The (15, 5) decomposition was much slower than the (9, 3) decomposition, since the time necessary to solve each subproblem grew exponentially. Still, it found slightly better solutions, and the processing time was reasonable (although it varied from problem to problem) if the heuristic dominance property and second lower bound were used.

Table 3.21. Performance of heuristics.

Problem Set	Shortest Waste	(9,3)	(15,5)	Genetic Algorithm
FT151	1.095	1.005	-	1.010
FT301	1.067	1.000	0.992	1.006
FT302	1.066	1.000	0.995	1.005
FT304	1.031	1.000	0.994	1.005

Notes: Performance measured against optimal solution for FT151. Against decomposition (9,3) for 30-job problems. All performances are average ratios over 10 problems. Performance of genetic algorithms averaged over three runs of 3000 individuals.

Although the initial lower bounds for the 30-job problems were not good, we were able to improve them using the branch-and-bound tree to show that the decomposition (9, 3) heuristic was within ten percent of the optimal flowtime.

The Shortest Waste heuristic found good solutions very quickly and generally performed better than other dispatching rules. On the 15-job problems, the genetic algorithm was not an effective heuristic, since it required more computation time than the branch-and-bound and could not always find optimal solutions.

On the 30-job problems, the problem space genetic algorithm found solutions better than Shortest Waste and as good as the decomposition heuristic. The computation time was slightly longer for a 3000-individual search than for a (15, 5) decomposition, but a 1000-individual search was much shorter and found solutions with little increase in total flowtime. The exponential nature of genetic search is exhibited in Figure 3.12. (In other testing, we found that the genetic algorithm was not as effective when using a simple Earliest Release Date rule to create sequences).

All programs were run on a 386 PC. Decreases in times were achieved when the programs were run on a 486 PC, and further decreases could be achieved on a more powerful machine. Except for the 30-job branch-and-bound (which we could not solve), we do not consider processing times to be a significant obstacle. The numbers in Table 3.22 are offered only for comparison purposes.

Table 3.22. Typical computation times

FT302.1
Decomposition (9,3): 17.8 seconds
Decomposition (15,5): 338.67 seconds
1000-individual GA: 134.62 seconds
3000-individual GA: 350.37 seconds
Shortest Waste: < 0.1 seconds

FT151.1
Branch-and-bound: 10.71 seconds
Decomposition (9,3): 0.8 seconds
3000-individual GA: 149.24 seconds
Shortest Waste: < 0.1 seconds

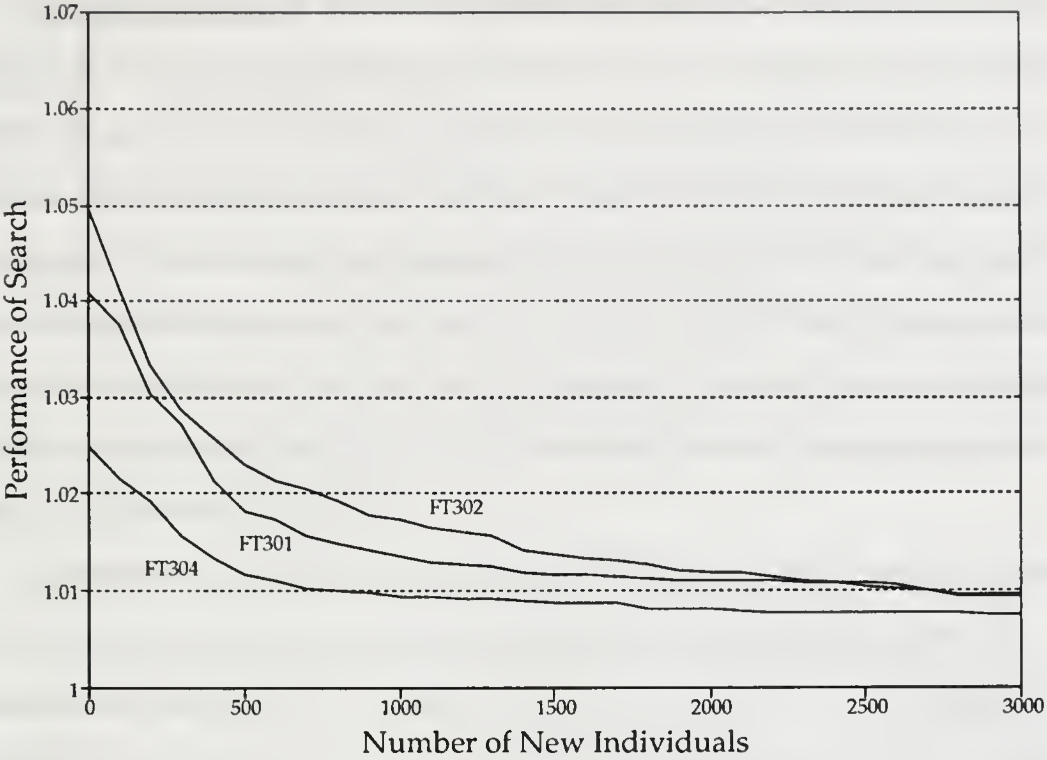


Figure 3.12. Performance of Shortest Waste Genetic Algorithm Performance measured against decomposition (9,3).

3.4.6 Special Case

In this section we consider a particular special case of the problem which may be useful in certain manufacturing situations. We will assume that there exist exactly two job classes and that the job processing times are equal within each class.

Specifically, we study the following instance: $p_j = p$ for all J_j in G_1, $p_j = q$ for all J_j in G_2. Since all of the jobs in a class have identical processing times, we may order them by ERD. We note here that all of the release times are integer.

We will describe a pseudo-polynomial dynamic program to solve the problem, a special case of the strongly NP-complete FTSRD problem.

Dynamic programming. We can use a dynamic program for two reasons: there are only two classes, and we have an ordering for the jobs in each class. According to Monma and Potts (1989), this *ordered batch scheduling problem* can be solved in pseudo-polynomial time. The following dynamic program interleaves the classes. The state variable in the dynamic program corresponds to a partial schedule that consists of the first i_1 jobs from G_1 and the first i_2 jobs from G_2 and that ends before or at a specific time with a job from a specific class (if it is cheaper to end sooner, that schedule should take precedence). At each point in the state space, we will measure the total flowtime of the scheduled jobs. The recursion determines the best partial schedule to which we should add the specified job.

Algorithm 3.3. Let $f(t, a, i_1, i_2)$ be the minimum flowtime of a partial schedule where the last jobs ends at or before time t, there are i_1 jobs from G_1 and i_2 jobs from G_2, and the last scheduled job is from G_a. $t = 0, \dots, R$. $a = 1, 2$. $i_1 = 0, \dots, n_1$, $i_2 = 0, \dots, n_2$. Renumber the jobs so that $j \leq n_1$ if J_j is in G_1, $r_1 \leq \dots \leq r_{n1}$, and $j > n_1$ if J_j is in G_2, $r_{n1+1} \leq \dots \leq r_n$. R is some upper bound on the makespan of a schedule. We can find one such R by scheduling all jobs in ERD order, performing a class setup in front of every job. We also have an upper bound: $R \leq \max \{r_j\} + \sum p_j + n_1 s_{21} + n_2 s_{12}$.

Initialization:

$f(t, a, i_1, i_2) = \infty$ if $t < 0$.

$f(t, 1, 0, i_2) = \infty$ for all t and for $i_2 > 0$.

$f(t, 2, i_1, 0) = \infty$ for all t and for $i_1 > 0$.

$f(t, 1, 1, 0) = \infty$ for $t < \max \{s_{01}, r_j\} + p$, if J_j is the first job in G_1 ($j = 1$).

$f(t, 1, 1, 0) = \max \{s_{01}, r_j\} + p$ for $t \geq \max \{s_{01}, r_j\} + p$, if J_j is the first job in G_1 ($j = 1$).

$f(t, 2, 0, 1) = \infty$ for $t < \max \{s_{02}, r_j\} + q$, if J_j is the first job in G_2 ($j = n_1 + 1$).

$f(t, 2, 0, 1) = \max \{s_{02}, r_j\} + q$ for $t \geq \max \{s_{02}, r_j\} + q$, if J_j is the first job in G_2 ($j = n_1 + 1$).

Iteration: $(i_1 + i_2 > 1)$

$f(t, 1, i_1, i_2) = \min \{f(t-p, 1, i_1-1, i_2) + t, f(t-p-s_{21}, 2, i_1-1, i_2) + t, f(t-1, 1, i_1, i_2)\}$ if $t \geq r_j + p$, where $j = i_1$.

$f(t, 1, i_1, i_2) = \infty$ if $t < r_j + p$, where $j = i_1$.

$f(t, 2, i_1, i_2) = \min\{f(t-q, 2, i_1, i_2-1) + t, f(t-q-s_{12}, 1, i_1, i_2-1) + t, f(t-1, 2, i_1, i_2)\}$ if $t \geq r_j + q$, where $j = n_1 + i_2$.

$f(t, 2, i_1, i_2) = \infty$ if $t < r_j + q$, where $j = n_1 + i_2$.

Answer: the optimal total flowtime is $\min\{f(R, 1, n_1, n_2), f(R, 2, n_1, n_2)\}$.

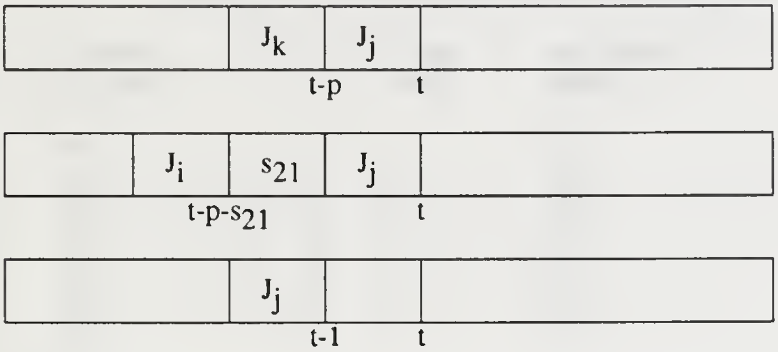


Figure 3.13. Schedules for iteration (J_j and J_k in G_1 and J_i in G_2).

In the iteration, the first term being considered for $f(t, 1, i_1, i_2)$ is the total flowtime of the partial schedule formed by adding the J_j ($j = i_1$) to a schedule ending at or before t-p with a job in G_1. The second term is the flowtime if J_j is added to a schedule ending at or before t-p-s_{21} with a job in G_2. The third term corresponds to a schedule that ends at or before t-1 with J_j. If $t < r_j + p$, there is no feasible schedule that ends with J_j; thus, $f(t, 1, i_1, i_2)$ is set to infinity. The iteration is similar for the $f(t, 2, i_1, i_2)$.

The effort for each point in the state space is constant. The effort for the entire program is thus proportional to the number of points in the state space. This is $O(Rn^2)$. Since R is bounded by a polynomial function of the problem data, the algorithm is pseudo-polynomial. The dynamic program can be implemented to find the optimal objective function value with memory requirements that are $O(Rn)$: If we are determining the values for points with $i_1 + i_2 = k$, any points in the space where $i_1 + i_2 < k - 1$ are ignored. If we define $f_k(t, a, i_1) = f(t, a, i_1, k - i_1)$, we need to keep only f_k and f_{k-1} at any time.

Test problems. We generated 70 test problems in order to test the dynamic program on a range of problem sizes. The data for the problem sets are summarized in Table 3.23 (10 problems in each set).

Table 3.23. Data about problem sets for special case.

Problem Set	Number of jobs	p	q	Class setup	Interarrival times
FT201	20	1	1	1	1-3
FT307	30	1	1	1	1-3
FT308	30	2	3	1	2-5
FT309	30	4	2	3	4-8
FT401	40	1	1	1	1-3
FT507	50	1	1	1	1-3
FT601	60	1	1	1	1-3

Results of special case dynamic program. The dynamic program finds optimal solutions in time that is nearly proportional to Rn^2 (see Table 3.24).

One drawback of the dynamic program is the amount of memory required to perform the algorithm; we were able to solve only 60-job problems. In addition, it requires significant processing time on a 386 personal computer. These problems can be overcome, however, since a larger computer could handle more memory (the amount required is $O(Rn)$), and would run more quickly.

Table 3.24. Results of dynamic program

Problem Set	Number of jobs	Average R	Average computation time
FT201	20	41.1	0.808
FT307	30	61.4	2.270
FT308	30	108.1	3.487
FT309	30	185.5	5.899
FT401	40	81.8	4.698
FT507	50	101.6	9.079
FT601	60	121.4	14.836

Extensions of special case. The dynamic program can be modified to solve any FTSRD problem where there exists a natural order for the jobs within each class. This includes problems where the jobs in each class have the same processing time (as we have discussed) or problems where all of the jobs have the same release date (order the jobs in each class by SPT). In any of these cases we have an ordered batch scheduling problem. If each class has a natural order, the dynamic program can be used to interleave these sequences since we have an ordered batch scheduling problem.

Recall from the example presented in Section 3.4.4 that matching processing times and release dates do not give us a natural order for a class.

3.4.7 Conclusions

In this research we have studied a computationally difficult class scheduling problem. The objective is to minimize the total flowtime of a set of jobs that have non-zero release dates. We examined a number of techniques to solve the problem, including a branch-and-bound search, look-behind dispatching rules, a decomposition heuristic, and a problem space genetic algorithm. We were interested in determining how this type of genetic algorithm can be used to find good solutions for another class scheduling problem.

Our results are as follows: While we did develop lower bounds and a number of dominance properties, our branch-and-bound approach was unable to solve any 30-job problems. The decomposition heuristic was a successful technique, locating solutions of high quality. The Shorest Waste heuristic could sometimes generate good solutions. However, by incorporating these rules in a genetic algorithm that searched the space of adjusted release dates, we could find much better solutions.

From these results we can conclude two things: For the one-machine class scheduling problem we call FTSRD both the problem space genetic algorithm and the decomposition heuristic can find good solutions in reasonable time. Additionally, look-behind rules may be

useful for job shop scheduling, especially on a bottleneck machine which undergoes class setups and where jobs continue to arrive while the machine is processing.

3.5 Chapter Summary

In this chapter we have presented the results of research into three one-machine class scheduling problems. We can make a number of observations about this research. 1. None of these three problems have been previously considered in the literature. 2. We have presented analytical results and developed extended heuristics for each of these problems. 3. All three problems are motivated by the semiconductor test area job shop environment, and the extended heuristics developed for these problems may be useful as dispatching rules in the general job shop scheduling problem. 4. Our problem space genetic algorithm is a robust approach, able to find good solutions over a variety of one-machine class scheduling problems, and should be applicable to other difficult combinatorial and scheduling problems.

CHAPTER 4
LOOK-AHEAD SCHEDULING PROBLEMS

In this chapter we discuss the second major area of this research. We study the problem of scheduling a machine that processes jobs headed for two different second-stage machines. We analyze this three-machine problem with three different objective functions: makespan, total flowtime, and number of tardy jobs. We first examine the complexity of the problem and then identify some lower bounds as well as some special cases that can be solved in polynomial time. We develop a number of heuristics that find good solutions to the problem. We also use a branch-and-bound technique to find optimal solutions.

4.1 Introduction

Job shop scheduling includes those scheduling problems in which different jobs may follow different routes through the shop. These problems are generally the hardest to solve optimally, since few properties of optimal schedules are known and the number of possible solutions explodes as the problems increase in size.

Because of the complexity of job shop scheduling, algorithms to find the optimal solution (in a reasonable amount of time) for even the simplest objective functions, e.g. makespan, do not exist. Recent research has shown that bottleneck-based techniques such as the shifting bottleneck algorithm (Adams, Balas, and Zawack, 1988) or bottleneck dynamics (see Morton, 1992, for example) can be successful at finding good schedules. Traditionally, however, researchers have studied (and schedulers have used) dispatching rules to order the jobs waiting for processing at a machine.

Normal dispatching rules consider only the jobs currently in the queue for the machine being scheduled. We define *look-ahead scheduling* as the ability of a sequencing procedure to

include information about the status of machines downstream in the flow, enabling it to make a better solution. Previous techniques that use this look-ahead idea include the work-in-next-queue and number-in-next-queue dispatching rules (Panwalker and Iskander, 1977) and the use of bottleneck starvation avoidance in shop floor control by Glassey and Petrakian (1989). Robinson *et al.* (1993) consider upstream and downstream information in scheduling semiconductor batch operations. Researchers have also studied lot release policies that look ahead to the status of the inventory in front of or arriving at a bottleneck; see for example, Wein (1988), Glassey and Resende (1988), and Leachman, Solorzano, and Glassey (1988). Other researchers have studied procedures that they called look-ahead scheduling (Koulamas and Smith, 1988; Zeestraten, 1990) but the problem setting or interpretation is different.

Consider two examples from the semiconductor test area. In the first, lots of two different products are processed through the same brand workstation. After brand, the lots require electrical testing, but the differences between the products indicate that the lots must be tested on different machines. Or consider the effect of re-entrant flows. The various lots waiting for processing at an electrical test workstation may be at one of two points in their route. At one point, a lots moves to brand after being tested (and it will return to test at some point in the future). At another, it moves to visual/mechanical inspection. In the first case, the brand workstation is sending lots to two different testers; in the second, the tester is sending lots to two dissimilar workstations. If one of the second-stage machines is a bottleneck, it seems clear that the sequencing of lots on the first-stage machine should try to maximize the efficient use of that bottleneck.

We can model this scenario with the following three-machine problem: There are three machines M_0, M_1, and M_2. Each job follows one of two different flows: M_0 - M_1, or M_0 - M_2. Thus, M_0 is feeding the other two machines. If one of these second-stage machines is a bottleneck because the total work required on that machine is the larger than that on the other machine, the sequencing of jobs on M_0 should have as a priority the proper feeding of that

machine. This idea of a bottleneck will not affect our analysis of the problem. We will, however, return to it for the heuristics and the empirical testing.

This problem, which could occur in any number of manufacturing environments, forms an interesting general flow shop problem and a subproblem of the job shop scheduling problem. As a flow shop problem, it is a simple model unlike the multi-machine problems previously discussed in the literature, although work has been done on flexible flow shops with multiple parallel machines at any given stage.

Moreover, the research into our three-machine problem may improve job shop scheduling in two ways: one, the solution procedures can be applied directly to the subproblem of scheduling machines near the bottleneck machine, and two, these techniques may be translated into good look-ahead dispatching rules for scheduling throughout the shop.

In this chapter we investigate three objective functions for this problem: the minimization of makespan, of total flowtime, and of the number of tardy jobs. We are concerned with the analysis of solutions to the problem and the development of heuristics which can be used to find good solutions.

The major contributions of this work include the proof that minimizing makespan is a strongly NP-complete problem, the identification of optimality properties and special cases that can be solved in polynomial time, and the development of an approximation algorithm.

The look-ahead scheduling problems under investigation are as follows (using the numbering given earlier):

4. Three-Machine Look-Ahead Scheduling: Makespan (3MLA-MS)
5. Three-Machine Look-Ahead Scheduling: Flowtime (3MLA-FT)
6. Three-Machine Look-Ahead Scheduling: Number of Tardy Jobs (3MNT)

In this chapter we will look at each of these objective functions. In Chapter 2 we discussed the research relevant to these problems. In the next section we start with the makespan objective. In Section 4.3 we look at minimizing the total flowtime, and Section 4.4 discusses work on minimizing the number of tardy jobs.

4.2 Minimizing the Makespan

As mentioned in the introduction, we are studying a problem that is a subproblem of the general job shop scheduling problem and is also a special case of the general flow shop problem. All analysis of the flow shop starts with Johnson (1954), who studied the minimization of makespan for two-machine flow shop problems and for some special three-machine flow shop problems. His famous algorithm starts jobs with the smallest first-stage tasks as soon as possible and jobs with the smallest second-stage tasks as late as possible.

Special cases of the flow shop makespan problem have been studied by a number of researchers, including Mitten (1958), Conway, Maxwell, and Miller (1967), Burns and Rooker (1975), and Szwarc (1977). Garey, Johnson, and Sethi (1976) proved that the general three-machine problem was NP-complete. Problems with release dates, preemption, precedence constraints, or more than three machines have also been studied.

In the flexible flow shop, more than one machine may be present at a particular stage. Heuristics for this type of problem are discussed by Wittrock (1988), Sriskandarajah and Sethi (1989), Gupta (1988), and Gupta and Tunc (1991). Lee, Cheng, and Lin (1992) study an assembly flow shop problem where each job consists of two subassembly tasks that are assembled in a third operation.

This three-machine problem is therefore closely related to problems previously studied, but the pre-assignment of the jobs to different second-stage machines gives this problem a special structure and leads to interesting results.

4.2.1 Notation

The following list describes the components of the problem and the notation used.

J_j	Job j, $j = 1, \ldots, n$.
M_0	The first-stage machine.
M_1, M_2	The second-stage machines.
H_1	The set of jobs that visit M_0 and then M_1.
H_2	The set of jobs that visit M_0 and then M_2.
p_{0j}	The first-stage processing time of J_j on M_0.
p_{ij}	The second-stage processing time of J_j on M_i, $i = 1$ or 2.

For a given schedule σ, we can calculate the following variables:

C_{0j}	The completion time of J_j on M_0.
C_{ij}	The completion time of J_j on M_i, $i = 1$ or 2.
C_j	$= C_{ij}$, the second-stage completion time of J_j.
C_{max}	$= \max \{C_j\}$, the makespan of the schedule.
$\sum C_j$	the total flowtime of the schedule.

Note that we will call a set of jobs that visit the same second-stage machine a *group*; thus, H_1 and H_2 are each a group of jobs. Each group has a *flow*. The flow for the jobs in H_1 is M_0 - M_1. The flow for the jobs in H_2 is M_0 - M_2. This section is concerned with the problem of minimizing, over all feasible schedules, the makespan of the jobs. We call this problem the Three-Machine Look-Ahead problem - Makespan (3MLA-MS).

4.2.2 Johnson's Algorithm

If we consider just one group and its corresponding flow, the problem of minimizing the makespan of the set of jobs that visit these two machines is the same as Johnson's two-machine flow shop problem. Johnson (1954) provided an optimality rule and an algorithm to solve the problem. If each job to be scheduled has task processing times a_j and b_j on machines one and two respectively, then his rule is as follows:

Johnson's Rule: J_i precedes J_j in an optimal sequence if $\min \{a_i, b_j\} < \min \{a_j, b_i\}$.

This rule is implemented in the following algorithm:

Johnson's Algorithm:

Step 1. Find the smallest unscheduled task processing time. (Break ties arbitrarily.)

Step 2. If this minimum is on the first machine, place the job in the first open position in the schedule. Else, place the job in the last open position in the schedule. Return to Step 1.

4.2.3 Permutation Schedules

In flow shop scheduling, a feasible schedule is called a permutation schedule if the sequence of jobs on each machine is the same. Thus, the sequence of jobs on the first machine uniquely identifies a schedule (assuming all tasks are started as soon as possible). In the three-machine look-ahead problem that we are studying, the two sequences on the second-stage machines consist form two disjoint sets of jobs. Thus, for our problem, we extend the idea of permutation schedules to include schedules where the relative order of two jobs in the same flow is the same at both stages.

It is known that considering permutation schedules is sufficient for finding optimal solutions for regular two-machine flow shop problems. Thus, for our three-machine look-ahead problems, it seems likely that permutation schedules will also be sufficient, since the problem contains two two-machine flows. We will show that this is indeed true.

Definition: A schedule is a *permutation schedule* if, for all J_j and J_i in H_1 (H_2), J_i precedes J_j on M_0 if and only if J_i precedes J_j on M_1 (M_2).

Theorem 4.1. For any regular performance measure, there exists a permutation schedule that is an optimal schedule.

Proof. First we will show that we can interchange two jobs that are not in the same order at both stages. Given a schedule σ where job J_j directly follows J_i on machine M_1 (or M_2) but J_j precedes J_i on machine M_0, move the M_0 task of J_j after the M_0 task of J_i.

If we look at Figure 4.1, we can observe that moving J_j causes all of the tasks (except J_i) on M_0 to start earlier, which does not delay any (and may expedite some) second-stage tasks. Now, since J_j will complete on M_0 when J_i did ($C'_{0j} = C_{0i}$), the processing of J_j on M_1 will not be delayed. Our interchange, therefore, does not increase the completion time of any job.

Thus, for any given schedule, we can create a corresponding permutation schedule by interchanging the first-stage tasks. None of the job completion times are increased by this construction. Indeed, some of the completion times may be decreased. Therefore, this permutation schedule has a better or equal performance on all regular measures (e.g. makespan, flowtime, number of tardy jobs, maximum lateness). Thus, it is sufficient to consider permutation schedules when trying to minimize these objective functions. QED.

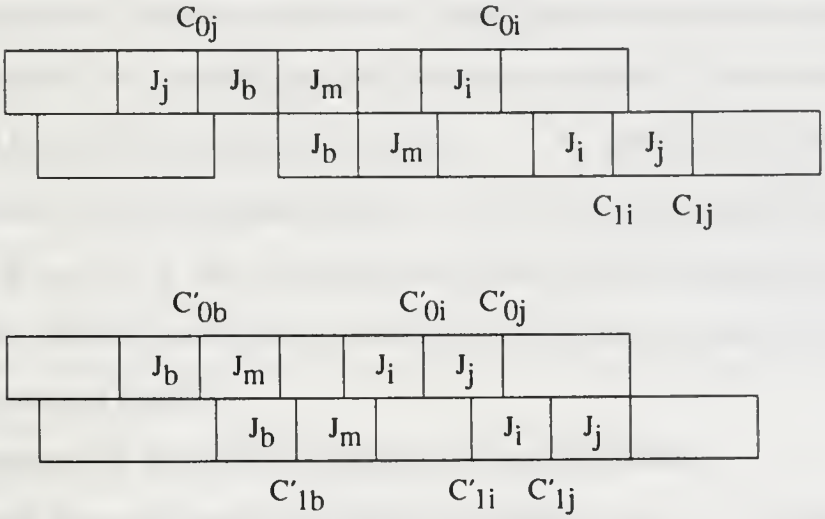


Figure 4.1. The exchange into a permutation schedule for M_0 and M_1. (Machine M_2 not shown.)

We would note here that a permutation schedule preserves the relative order of the sequence for each group. We will use the term *interleaving* for the process of combining two sequences to form a permutation schedule. We will show in Section 4.2.5 that if we are given sequences for each group, then there is a polynomial-time algorithm to find the interleaving that minimizes the makespan of the schedule created. However, the optimal second-stage sequences cannot be determined in polynomial time. In fact, the 3MLA-MS problem is strongly NP-complete, as will see in the next section.

4.2.4 NP-Completeness

In this section, we consider the complexity of the 3MLA-MS problem. Although other researchers (including Gonzalez and Sahni, 1978) have shown the NP-completeness of a number of small shop problems, we cannot determine the complexity of our problem from any of this previous work.

We will therefore prove that 3MLA-MS is strongly NP-complete, which will be done by transforming 3-Partition to 3MLA-MS. (Recall that it is sufficient to consider permutation schedules.) The 3-Partition problem can be stated as follows: Given a set of a_i, $i = 1, \dots, 3n$, and B, partition these elements into n sets $A_1, \dots, A_n$, where each set contains three elements and the sum of the three elements equals B. We will make the assumption that for all i, $1/4\ B < a_i < 1/2\ B$. (We can transform any problem without this property into one where it is true.) With this property, there will never be a set of two a_i or a set of four a_i where the sum of the elements equals B.

Theorem 4.2. 3MLA-MS is a strongly NP-complete problem.

Proof. Given an instance of 3-Partition, with a set of a_i, $i = 1, \dots, 3n$, and B (note all $a_i \geq 1$), create $4n$ jobs, $3n$ of which go from M_0 to M_2 and have processing times $p_{0i} = a_i$ and $p_{2i} = (B+1)a_i$, $i = 1, \dots, 3n$; let these jobs form a set W. The n remaining jobs that go from M_0 to M_1, $J_{3n+1}, J_{3n+2}, \dots, J_{4n}, p_{0k} = B^2$ for all k and $p_{1k} = (B+1)B$, $k = 3n, \dots, 4n\text{-}1$, and $p_{1,4n} = B$; let these jobs form a set X. The desired makespan is $M = n(B^2 + B) + B$.

Part 1. If there exists a partition $(A_1, \dots, A_n)$ such that for all a_i in A_k, $\Sigma\, a_i = B$, then the sequence on M_0 of $[A_1\, J_{3n+1}\, A_2\, J_{3n+2} \dots A_n\, J_{4n}]$ will yield a schedule where the completion time on M_1 will be M and the time on M_2 will be no greater than M. (See Figure 4.2.)

Part 2. Now, suppose there exists no partition. Consider any arbitrary permutation schedule σ. The sequence is composed of consecutive subsets of jobs from W and of jobs from X. Let A_j, $j = 1, \dots, n+1$ be these subsets of W, where A_j directly precedes the jth job from X, except for A_{n+1}, which is the set of jobs following the last job from X. Note that any of these A_j

may be empty. Let S_j equal the sum of the first-stage task processing times for the jobs in A_j. Let A_k be the first A_j such that S_k is not equal to B. Because there is no partition, there must exist one such A_k among $A_1, \ldots, A_n$.

If $S_k > B$, then the makespan on machine M_1 is delayed by the delay in the kth job in X: $MS(M_1) \geq S_1 + B^2 + \ldots + S_k + B^2 + (n - k)(B^2 + B) + B > kB + kB^2 + (n - k)(B^2 + B) + B = M$. (This assumes that all of the first k-1 jobs have long tasks on M_1, leaving n-1-(k-1) long tasks and one short task on M_1. If one of the first k-1 has the short task, then the makespan is even longer.)

If $S_k < B$, let $S = S_1 + \ldots + S_k < kB$ (and $\leq kB - 1$) and suppose J_m is the first job from W after the kth job from X. Then, the makespan on M_2 is postponed by J_m: $MS(M_2) \geq S + kB^2 + a_m + (B + 1)(nB - S) \geq S + kB^2 + a_m + nB^2 + nB - BS - S$. Including $-BS \geq -kB^2 + B$ implies $MS(M_2) \geq a_m + nB^2 + nB + B > M$. Thus, there exists no schedule with makespan less than or equal to M.

Part 1 and Part 2 of the proof show that there exists a partition if and only if there exists a schedule with the desired makespan. This implies that 3MLA-MS is a strongly NP-complete problem. QED.

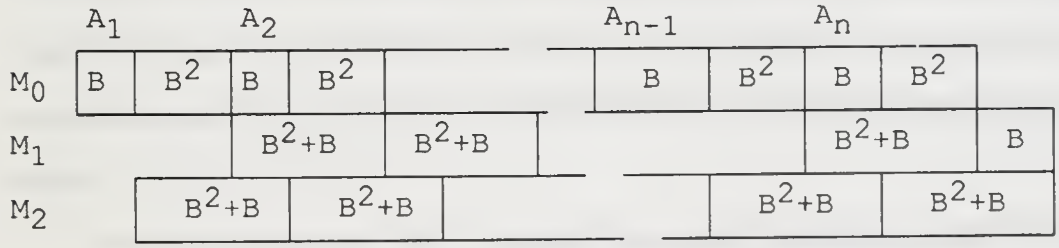


Figure 4.2. A schedule for the 3MLA-MS problem.

4.2.5 Makespan Optimality Conditions and Polynomially-Solvable Cases

This section discusses the optimal combination of two given sequences, lists a few properties of optimal schedules, and presents two special cases that have easily-found optimal solutions. We will see that we can easily form a permutation schedule by optimally interleaving

sequences for each group (Algorithm 4.1). Properties 4.1, 4.2, and 4.3 are dominance properties between jobs in the same group. Theorems 4.3 and 4.4 describe the special cases.

Because we need to consider only permutation schedules, we can create a single schedule for all three machines from a sequence of the jobs. That is, given a sequence on the first-stage machine we can create a sequence for each second-stage machine. This is done by considering the jobs in the order they appear on the first machine. However, there is no comprehensive rule for determining this sequence. Thus, we will spend some time on how this sequence should be constructed in certain situations.

Interleaving two sequences. Since the problem is NP-complete in the strong sense, heuristic methods for finding good solutions are justified. A very natural heuristic is to schedule the jobs in each group separately. For instance, each group can be scheduled by Johnson's rule. Then we can interleave these two sequences to form a solution to the original problem. The process of combining two sequences to achieve the minimal makespan is called *optimally interleaving*. This is the best combination of these two sequences. Note, however, that the best combination of the two Johnson sequences may not be an optimal solution; the optimal solution may be some combination of sub-optimal solutions to each subproblem. In Section 4.2.9 we present an example where optimally interleaving the Johnson sequences for each group is not optimal.

So, while interleaving the Johnson sequences is a natural heuristic that works well (see Section 4.2.8), we may wish to try other heuristics to the subproblems before interleaving. These are discussed in Section 4.2.7. In this subsection we will describe how two sequences should be interleaved.

Now, suppose we are given two sequences σ_1 and σ_2, one sequence for each group (we have found solutions for each of the subproblems), and we want to find the minimal makespan that can be achieved by combining these two sequences. The following observations will lead us to an algorithm for doing so.

Define C as a makespan that we wish to achieve. To minimize the makespan given the two sequences we need to find the minimal C for which we can find a feasible schedule. We can schedule the tasks on the second-stage machines in the order given by σ_1 and σ_2 and as late as possible, so that the last task on each machine ends at C. In a feasible schedule, the first-stage task for each job has to complete on M_0 before the second-stage task can begin. We need to determine if there is some ordering of the tasks on M_0 so that each task finishes on-time (with respect the second-stage task).

For each job J_j the start time of the second-stage task can be used as the due date d_j for the corresponding first-stage task. We can find a sequence where each first-stage task finishes on-time ($C_{0j} \leq d_j$) if and only if we can find a sequence where the maximum lateness is less than or equal to zero. If we wish to minimize the maximum lateness of the first-stage tasks, we should order them by EDD (Earliest Due Date), according to Jackson (1955). The schedule created is an interleaving of the two given schedules: if J_i precedes J_j in σ_k, then $d_i < d_j$, and J_i will precede J_j on M_0.

Each due date $d_j = C - t_j$, where t_j is the sum of the second-stage processing times of J_j and the jobs J_k that follow J_j on the second-stage machine. See Figure 4.3.

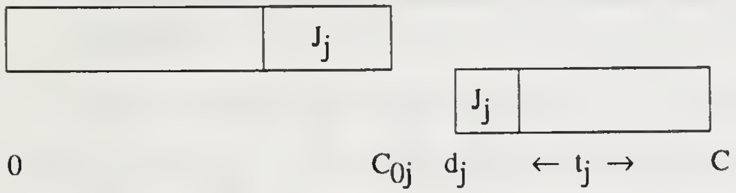


Figure 4.3. The variables associated with J_j (second-stage started late).
Note: Other second-stage machine not shown.

Since the EDD sequence corresponds to sequencing the tasks in decreasing order of t_j, the sequence of jobs is the same for all values of C, including the optimal one. Therefore, we can find the optimal makespan from the feasible schedule with the jobs in this order and all tasks starting as soon as possible. This is the optimal interleaving of two given sequences, described in Algorithm 4.1.

Algorithm 4.1 (Optimal Interleaving): Given a sequence σ_1 for the jobs in H_1 and a sequence σ_2 for the jobs in H_2, perform the following steps to yield a schedule with the minimal makespan:

Step 1. For each J_j in H_1, define A_j as the set of jobs (not including J_j) that follow J_j in σ_1. Then $t_j = p_{1j} + \Sigma_{Aj} p_{1k}$.

Step 2. For each J_j in H_2, define A_j as the set of jobs (not including J_j) that follow J_j in σ_2. Then $t_j = p_{2j} + \Sigma_{Aj} p_{2k}$.

Step 3. Schedule the jobs on M_0 in decreasing order of the t_j, starting at time zero, and start all second-stage tasks as soon as possible.

Note that this algorithm takes O(n) effort, since each group is already in decreasing order of the t_j, and forming the schedule is only combining the two sequences without changing the relative orderings.

In Example 4.1 we perform Algorithm 4.1 on a problem with five jobs and given sequences for each group. We are given sequences [$J_1 J_3 J_2$] and [$J_4 J_5$] for each group. After computing the t_j, we form the interleaved sequence [$J_1 J_3 J_4 J_5 J_2$]. The corresponding schedule has a makespan of 11 (see Figure 4.4).

Example 4.1. Given the following five jobs and the two sequences [$J_1 J_3 J_2$] and [$J_4 J_5$]:

J_j	H_i	p_{0j}	p_{ij}	t_j
J_1	H_1	2	4	9
J_2	H_1	1	1	1
J_3	H_1	2	4	5
J_4	H_2	2	1	4
J_5	H_2	1	3	3

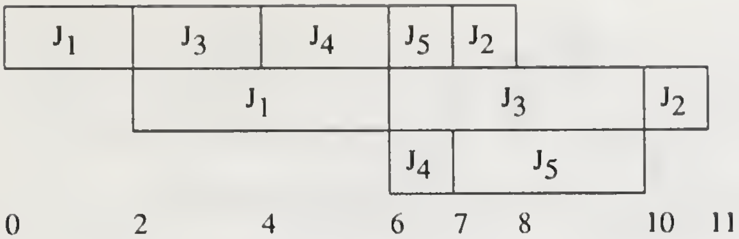


Figure 4.4. Optimal schedule for Example 4.1.

Properties of optimal solutions. In general, we still do not know how to construct the sequences on M_1 and M_2. These subproblems are what is difficult about this problem. Johnson's rule, which is optimal for the two-mchine flow shop, is not always applicable for the 3MLA-MS problem. Of some help, however, are a number of dominance properties for jobs in the same group that limit the number of sequences that need to be considered in searching for the optimal. The first property establishes a precedence relationship between two jobs that is a restrictive form of Johnson's rule. The other two relate two jobs that are processed consecutively on some machine.

Property 4.1. **(Absolute Dominance)** If both J_i and J_j are in H_k and $p_{0i} \le p_{0j}$ and $p_{ki} \ge p_{kj}$, then J_i should precede J_j. (If both equalities hold, then the jobs are obviously interchangeable.)

Proof. Suppose we have a schedule σ such that the inequalities above hold and J_j precedes J_i. Without loss of generality, suppose both jobs are in H_1 and that all of the second-stage tasks are started as late as possible. Then it is easy to see that exchanging J_i and J_j on both machines (see Figure 4.5) does not increase the makespan. Thus, it is sufficient to consider schedules where J_i precedes J_j. QED.

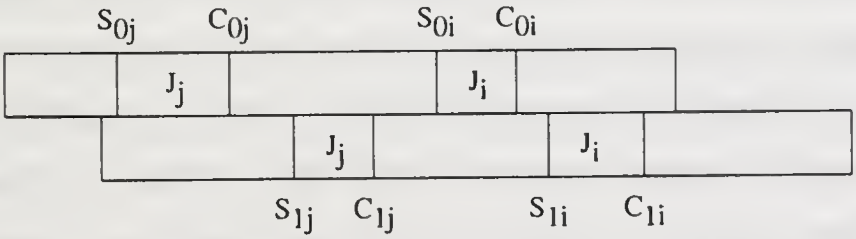


Figure 4.5a. J_i and J_j on M_0 and M_1. (Machine M_2 not shown.)

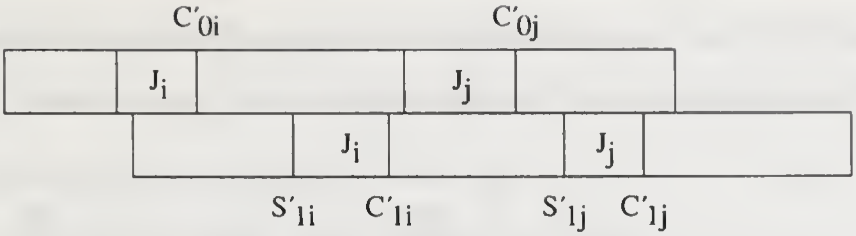


Figure 4.5b. The exchange of J_i and J_j on M_0 and M_1. (Machine M_2 not shown.)

Property 4.2. (Consecutive Dominance) If both J_i and J_j are in H_k and are processed consecutively on M_0, they should be sequenced by Johnson's rule (i.e., if min $\{p_{0i}, p_{kj}\}$ < min $\{p_{ki}, p_{0j}\}$, then J_i should immediately precede J_j).

Proof. The fact that both jobs are processed consecutively on both machines implies that Johnson's Rule applies to them. If the jobs are not sequenced according to Johnson's Rule, they can easily be exchanged, and the makespan of the schedule is not increased by the switch. QED.

This property is stated for two jobs from the same group processed consecutively. It can be extended to a set of three of more consecutive jobs on M_0 from the same group:

Corollary 4.1. (Batch Dominance) The jobs in each batch of a schedule should be ordered by Johnson's Rule, where a batch is defined as a set of consecutive jobs on M_0 from the same group.

Property 4.3. (Small Second-stage Dominance) Suppose J_i and J_j are in H_k and are processed consecutively on M_k. If $p_{0i} \geq p_{ki}$, $p_{0j} \geq p_{kj}$, and $p_{ki} > p_{kj}$, then J_i should precede J_j.

Proof. Without loss of generality, suppose J_i and J_j are in H_1. Consider a schedule where the two groups have been optimally interleaved and the second-stage tasks are started as late as possible. Then, if there are any jobs between J_i and J_j on M_0, they must be from H_2, and the start times of these jobs on M_2 are between the start times of J_i and J_j on M_1. (See Figures 4.6a and 4.6b.) Suppose J_j precedes J_i. Let us interchange these jobs on both M_0 and M_1 and check that we still have a feasible schedule without increasing the makespan. Let y be the old start time of J_j on M_1. Let z be the old completion time of J_i on M_0. For J_j, the new completion time on M_0

equals z. Since J_i can be on only one machine at a time, $z \le y + p_{1j}$, the old start time of J_i on M_1. Thus, $z < y + p_{1i}$, the new start time of J_j on M_1, and the schedule is still feasible for J_j.

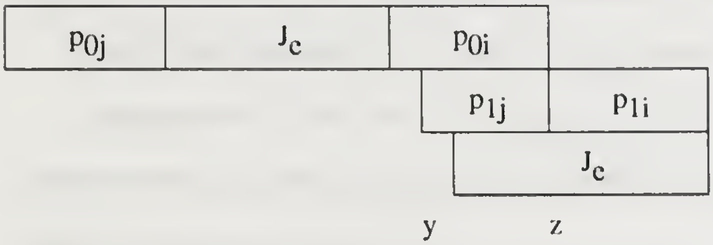


Figure 4.6a. (second-stage tasks started late).

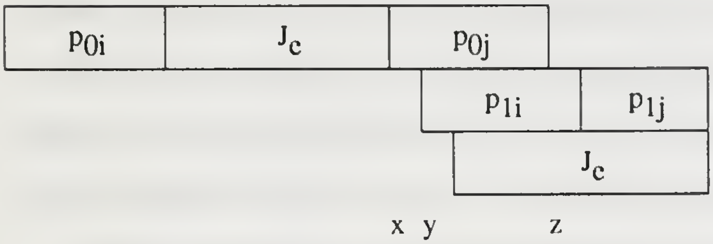


Figure 4.6b. (second-stage tasks started late).

Now, let x be the new start time of J_j on M_0. Thus $y \ge z - p_{1j} \ge z - p_{0j} = x$. Now, for J_i and for all jobs J_e between J_j and J_i, the new completion time is less than x and thus y and thus the start time of their second-stage tasks. We can therefore switch J_j and J_i without increasing the makespan of the schedule. QED.

Corollary 4.2. If for some H_k, $p_{0j} \ge p_{kj}$ for all J_j, then the jobs in H_k should be ordered by LPT on the M_k tasks.

We note that if Corollary 4.2 is true for both H_1 and H_2, we have the first special case mentioned below (Theorem 4.3).

<u>Polynomially-solvable special cases</u>. Like other three-machine flow shop problems, the 3MLA-MS problem has special cases that can be solved in polynomial time. Theorems 4.3 and 4.4 present such special cases.

Theorem 4.3. If $p_{1k} \le p_{0k}$ for all J_k in H_1 and $p_{2j} \le p_{0j}$ for all J_j in H_2, then the optimal solution can be found by ordering the jobs in each group by their second-stage processing times, longest processing time first, and optimally interleaving the two sequences.

Proof. This is true because Corollary 4.2 holds for both groups. QED.

Theorem 4.4. If, for some H_i, $\min \{p_{0j} : J_j \in H_1 \cup H_2\} \ge \max \{p_{ij} : J_j \in H_i\}$, then the optimal schedule can be found by sequencing the jobs in each group by Johnson's Rule and optimally interleaving these sequences.

Proof. By the given, we know that the conditions of Corollary 4.2 hold for the jobs in H_i. Thus, the jobs in this group should be ordered by longest second-stage task processing time first. For these jobs, this sequence is the same as the sequence given by Johnson's rule. Now, if we can determine the optimal ordering of the jobs in H_k, the other group, we can interleave the groups to derive an optimal schedule. We will show that these jobs should be ordered by Johnson's rule.

Consider an optimally-interleaved schedule where J_h is processed immediately before J_j on M_k, and where the jobs are not ordered by Johnson's rule. That is, $\min \{p_{0j}, p_{kh}\} < \min \{p_{0h}, p_{kj}\}$.

If there are no jobs from H_i between J_h and J_j on M_0, then Property 4.2 implies that the jobs should be interchanged.

Else, let J_m be the job from H_i processed just before J_j. If J_j is not the last job on M_0, then we can move J_j to immediately after J_h. Because the jobs that were between J_h and J_j have short second-stage tasks, delaying these jobs does not delay the processing of any successive jobs. Now J_h and J_j are consecutively processed on M_0, and Property 4.2 implies that the jobs should be interchanged.

Finally, if J_j is the last job on M_0 (and M_k), then the fact that the schedule is optimally interleaved implies that $p_{kj} \le p_{im}$, since J_m is the last job on M_i. By the given, p_{im} is less than or equal to p_{0h} and p_{0j}. Thus, $\min \{p_{0j}, p_{kh}\} < \min \{p_{0h}, p_{kj}\}$ implies that $p_{kh} < \min \{p_{0h}, p_{kj}\}$ (since $p_{kj} \le p_{0j}$) and consequently $p_{kh} < p_{0h}$ and $p_{kh} < p_{kj}$. Property 4.3 implies that J_j should precede J_h.

We have thus shown that the jobs in H_k should be ordered by Johnson's rule. This gives us an optimal ordering for H_k which can be interleaved with H_i to solve the problem. QED.

4.2.6 Branch-and-Bound Algorithm

In this section we will identify three lower bounds on the makespan; we will take the maximum of these to form our overall lower bound. We will also discuss the use of a branch-and-bound technique for finding optimal solutions for the 3MLA-MS problem. We will use the overall lower bound in the analysis of the worst-case performance of an approximation algorithm (Section 4.2.9). We will begin by discussing the three component lower bounds.

For the first component bound, consider only the jobs in H_1. Order these jobs using Johnson's rule and determine for each J_j a completion time C_{1j} on M_1. Then, $LB_1 = \max \{C_{1j}\}$. Similarly, $LB_2 = \max \{C_{2j}\}$.

The third component bound takes into account that all of the jobs use M_0. We relax the problem by dropping the second-stage assignments of the jobs and allowing an infinite number of second-stage machines. In fact, however, we only need n second-stage machines, one for each job. If each job has a separate second-stage machine, then the minimal makespan, which will be a lower bound on the optimal makespan for the original problem, is the maximum sum of first-stage completion time plus second-stage task processing time. By an argument similar to that of Section 1.4, we can find the minimal makespan by sequencing the jobs by their second-stage task processing times, longest first. We schedule the jobs in this order on M_0. The second-stage completion time of any job J_j in H_i is $C_{0j} + p_{ij}$. Therefore, $LB_3 = \max \{C_{0j} + p_{ij}\}$. Note that LB_3 will be greater than $\Sigma\, p_{0k}$, since there exists some job J_j that has $C_{0j} = \Sigma\, p_{0k}$.

Our lower bound for a given problem (hereafter referred to by the variable LB) will be the maximum of these three lower bounds for the makespan: (i) LB_1, the minimum possible makespan of the jobs in set H_1, (ii) LB_2, the makespan of the jobs in set H_2, and (iii) LB_3, the minimal makespan if there exist an infinite number of second-stage machines. That is, $LB = \max \{LB_1, LB_2, LB_3\}$.

In a standard branch-and-bound algorithm, each node will consist of a partial schedule of jobs. From a node, we exclude certain branches using the above dominance properties and create a lower bound using straight-forward extensions of the above lower bounds.

In order to help prune branches from the search tree, we will use the optimal properties that we developed in Section 4.2.5. We will also use a dominance property (Property 4.4) that is more dependent upon the current makespan of each machine (where t_i is the makespan of M_i).

Property 4.4. For a given partial schedule, if J_j is unscheduled and in H_k (where H_m is the other group), $t_k \geq \sum p_{0i}$ (the sum over all jobs), and $t_m < \sum p_{0i}$, then J_j should not precede any job from H_m.

It is easy to show that if, in any schedule constructed from this partial schedule, J_j does precede any of the jobs from H_m, we can move J_j to the last position without increasing the makespan.

We used the branch-and-bound algorithm to solve a number of problems. On the set of 15-job problems, the running time was usually less than one second (on a 386 PC), although it was much greater for two problems, one of which had a lower bound that was not tight. See Table 4.1.

Table 4.1. Statistics on branch-and-bound procedure for LA154.

Problem	Lower Bound	Optimal	Time	Nodes
1	2942	2942	0.11	38
2	2080	2080	0.11	39
3	3138	3138	0.16	47
4	3122	3122	0.11	37
5	1956	1956	0.16	56
6	2034	2034	0.22	51
7	2187	2187	0.17	57
8	1919	1919	0.17	71
9	1911	1911	73.92	61916
10	1945	1956	637.30	546654

Note: Time measured in seconds.

4.2.7 Heuristics

Since 3MLA-MS is a strongly NP-complete problem and the branch-and-bound procedure may occasionally take too long to find the optimal solution, the use of approximation algorithms is a preferable alternative. Our discussion so far has led us one very natural heuristic, the interleaving of the Johnson sequences. We will also introduce another combination that will be of use later.

Johnson Interleaved. Order the jobs in each group using Johnson's rule. Optimally interleave them using Algorithm 4.1.

Merged Johnson. Order the jobs in each group using Johnson's rule. Select the group with the smallest total task processing time on M_0 (if the totals are equal, pick one arbitrarily). Start the schedule with all of the jobs from this group. Follow them with the jobs from the other group.

4.2.8 Empirical Results

In this section we report on the empirical testing of our heuristics. We discuss the problem sets generated, our methodology, and our results.

In order to study the scheduling of a bottleneck machine, consider one of the second-stage machines, say M_1, as a bottleneck operation. As a bottleneck, the workload of M_1 should be greater than that of M_0 or M_2. Therefore, we will construct problems where the total processing time on M_1 is likely to be the largest.

Three problem sets (LA154, LA304, LA504) were created using uniform distributions to generate processing times. The mean fraction of jobs in each group and the mean processing times were set such that M_1 would have more work to do than M_2. These problems had 15, 30, and 50 jobs. There were ten problems in each set.

Table 4.2. Characteristics of 3MLA-MS problem instances.

Set	Jobs	Average number of jobs in H_1	M_0 times (range)	M_1 times	M_2 times
LA304	30	15	20-60	40-120	30-90
LA154	15	5	80-160	240-480	120-200
LA504	50	37	30-60	40-80	60-120

Another problem set (LA201) was created based upon the problem structure used in the NP-completeness proof. Ten sets of 15 random integers ranging from 1 to 10 was created (with an adjustment to insure that the total processing time was divisible by 5), and a 20-job problem was created from each set of these a_j using the construction in Theorem 4.2. For this problem we knew a good lower bound, although we determined later that two of the embedded 3-Partition problems had no solution and thus the lower bound could not be achieved.

For each problem, our lower bound on the makespan was the maximum of the three component bounds (see Section 4.2.6). The best component bound depended upon which machine had the largest workload for any problem. Since M_1 generally had the most processing time, the M_0 - M_1 bound was usually greatest.

For the 15-job problems, we were able to calculate the value of the optimal solution. For the other uniform problems we were able to find heuristic solutions that achieved the lower bound, implying that our lower bounds were tight and that we had found optimal solutions.

On the 20-job hard problems, we could solve the imbedded 3-Partition problem. For 8 of the 10 problems, there does exist a partition and thus we know the optimal makespan. For the other two, we do not know the optimal, although we came very close with the heuristics.

The results of the Johnson Interleaved and Johnson Merged heuristics on minimizing the makespan are shown in Table 4.3. Performance is the relative deviation from the optimal (if known) or the lower bound. We observed that the Johnson Interleaved algorithm found near-optimal or optimal solutions for all of these problems. The performance of the Johnson Merged algorithm varied from good on the 30- and 50-job problems to poor on the 20-job problems. The Johnson Interleaved heuristic outperformed a number of other sequencing rules, and these results were consistent with testing on a number of other problem sets with different problem structures.

Table 4.3. Makespan summary table for 3MLA-MS problem. Peformance is relative deviation from optimal or lower bound.

Set	Johnson Interleaved	Johnson Merged
LA 154	0.50	2.44
LA 304	0.00	0.35
LA 504	0.00	0.40
LA 201	0.42	23.14

4.2.9 Heuristic Error Bounds

It is often possible to evaluate a heuristic by analyzing its worst-case behavior. Heuristics that minimize the makespan of scheduling problems are especially open for this kind of analysis due to the simplicity of the objective function. Since the Johnson Interleaved heuristic performed the best in our empirical study, we are most interested in determining the worst-case performance of this procedure.

We will begin our analysis with the Merged Johnson procedure. For this bound we will make use of the lower bounds derived in Section 4.2.6. Recall that our lower bound $LB = \max\{LB_1, LB_2, LB_3\}$.

Theorem 4.5. For a given instance of 3MLA-MS, the schedule created by the Merged Johnson heuristic has a makespan less than 3/2 LB.

Proof. Without loss of generality suppose that $\Sigma_{H1}\, p_{0j} \le \Sigma_{H2}\, p_{0j}$. Now, $LB_3 > \Sigma\, p_{0j}$, the sum over all of the jobs, implying $\Sigma_{H1}\, p_{0j} < 1/2\ LB_3 \le 1/2\ LB$. LB_k be the minimal makespan achievable by the jobs in H_k, found by scheduling the jobs using Johnson's rule. The makespan of the scheduled created using the Merged Johnson heuristic is $\max\{LB_1, \Sigma_{H1}\, p_{0j} + LB_2\}$. This is less than $\max\{LB, 1/2\ LB + LB\} = 3/2\ LB$. QED.

Now, let us consider the Johnson Interleaved heuristic. Since this algorithm combines the two Johnson sequences optimally, its performance cannot be worse than that of the Merged Johnson heuristic. Thus we can make the following statement.

Corollary 4.3. For a given instance of 3MLA-MS, the schedule created by the Johnson Interleaved heuristic has a makespan less than 3/2 LB.

Theorem 4.6. The maximum error of the Johnson Interleaved algorithm relative to the optimal makespan is one-half.

Proof. If C^* is the value of the optimal makespan and C_{JI} the makespan of the schedule created by the Johnson Interleaved algorithm, $LB \le C^* \le C_{JI} < 3/2\ LB \le 3/2\ C^*$. Thus, $(C_{JI} - C^*) / C^* < 1/2$. QED.

Let us make a few observations about our algorithm. First, the error bound can be extended to the case where there are $m > 2$ groups (each group with a different flow to a different second-stage machine). The maximum relative error of the Johnson Interleaved algorithm is $1 - 1/m$ in this case.

Second, the Johnson Interleaved algorithm interleaves the groups by looking ahead to the future workload of the second-stage machines (the sum of the remaining second-stage task processing times).

Finally, our error bound of one-half is tight. In the following problem instance, the bound is achieved in the limit.

Example 4.2. Given n, let C^* be $2n + 1$. For H_1, construct jobs $J_1, \ldots, J_n$ with the following characteristics:

$i = 1, \ldots, n - 1$:	$p_{0i} = 1$	$p_{1i} = 1$
$i = n$:	$p_{0n} = 1$	$p_{1n} = n$.

Construct in H_2 jobs $J_{n+1}, \ldots, J_{2n}$ with similar characteristics:

$i = n + 1, \ldots, 2n - 1$:	$p_{0i} = 1$	$p_{1i} = 1$
$i = 2n$:	$p_{0,2n} = 1$	$p_{1,2n} = n$.

Now, since ties can be broken arbitrarily, each group is already ordered by Johnson's rule (we could force this ordering by subtracting a small amount from the appropriate first-stage tasks). An optimal schedule can be achieved by interleaving the sequences $[\, J_n J_1 \ldots J_{n-1} \,]$ and

[$J_{2n} J_{n+1} \dots J_{2n-1}$]. The makespan of such a schedule is $C^* = 2n + 1$. The interleaving of the Johnson sequences yields a schedule with makespan $3n$, as does merging the Johnson sequences. As n goes to infinity, the ratio of each of these makespans to C^* goes to 1.5. See Figures 4.7, 4.8, 4.9 ($J_x = J_{10}$) for the optimal, interleaved, and merged schedules with $n = 5$.

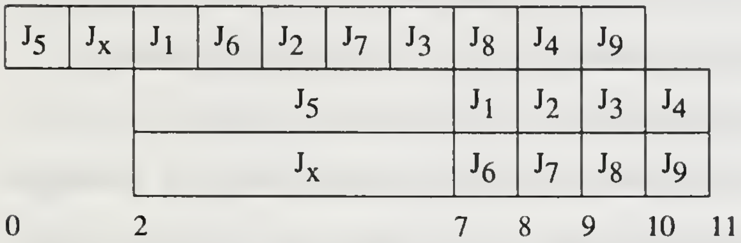


Figure 4.7. Optimal schedule (second-stage tasks started late).

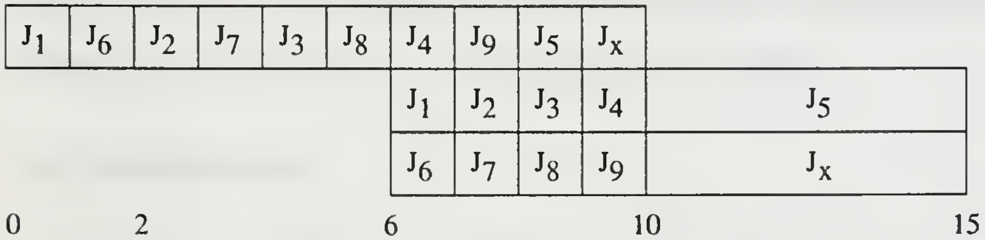


Figure 4.8. Interleaved Johnson schedule (second-stage tasks started late).

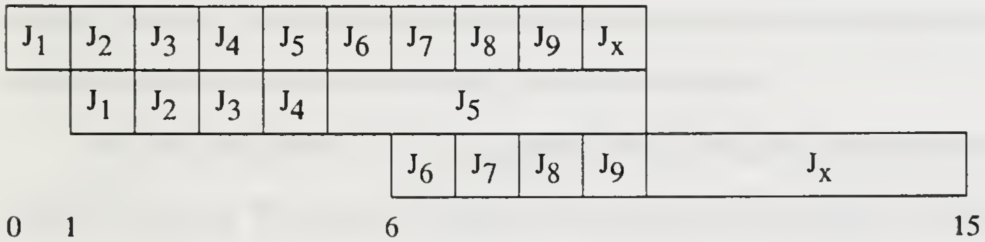


Figure 4.9. Merged Johnson schedule.

4.3 Minimizing the Total Flowtime

In this section we discuss the Three-Machine Look-Ahead problem with the objective of minimizing the total flowtime (3MLA-FT). We know that minimizing total flowtime in a two-machine flow shop is an strongly NP-complete problem (Garey, Johnson, and Sethi, 1976). Since

that problem is a special case of 3MLA-FT, 3MLA-FT must also be a strongly NP-complete problem. Hence, we will investigate optimality conditions, lower bounds, and heuristics. Since total flowtime is a regular objective function, only permutation schedules need be considered for optimality (see Section 4.2.3); thus, a sequence of jobs for M_0 corresponds to a unique schedule on all three machines. The problem notation is the same as that for the 3MLA-MS problem.

A number of researchers have studied the problem of minimizing the total flowtime in a flow shop. This work includes lower bounds, branch-and-bound algorithms, and heuristic approaches. See Ignall and Schrage (1965) for a branch-and-bound and Ahmadi and Bagchi (1990) for improved lower bounds. Szwarc (1983) studies special cases, and Van de Velde (1990) presents a Lagrangian relaxation. Krone and Steiglitz (1974), Kohler and Steiglitz (1975), Miyazaki, Nishiyama, and Hashimoto (1978), and Miyazaki and Nishiyama (1980) all present heuristic approaches. Ahmadi et al. (1989) includes batch processing.

4.3.1 Total Enumeration

Initially, we wanted the compare the difficulty of finding near-optimal solutions for 3MLA-FT to the difficulty of finding near-optimal solutions for 3MLA-MS, where we had very good results. One way in which we could compare the two problems was to compare the range of the objective functions over the domain of all permutation schedules.

For a nine-job problem, a total enumeration of the permutation schedules includes over 300,000 sequences. We picked an arbitrary instance, created each schedule, and measured the makespan and total flowtime of each schedule. This procedure yielded a distribution of makespans and total flowtimes over the set of sequences.

The primary result is that 18.5% of all sequences have makespans within 6.7% of the optimal makespan, but less than one quarter of one percent of all sequences have flowtimes within 6.7% of the optimal flowtime. (See Table 4.4 and Figure 4.10.) Because there are much fewer schedules that are near-optimal, we can infer that the problem of minimizing the total flowtime is a much harder problem than the problem of finding the minimum makespan. Similar

results hold for another nine-job problem and a fifteen-job problem where a random sample of the sequences were examined.

Table 4.4. Distribution of schedule makespans and flowtimes.

Deviation from optimal makespan	Percent of schedules	Deviation from optimal flowtime	Percent of schedules
3.3 %	5.65 %	3.7 %	0.05 %
6.7	18.47	6.7	0.23
10.0	30.13	14.2	2.05
13.3	49.07	21.6	6.85
16.7	62.17	29.1	15.83
20.0	75.41	36.6	29.18
23.3	86.24	44.0	45.37
26.7	92.31	51.5	62.31
30.0	96.43	59.0	77.28
33.3	98.10	66.4	88.65
40.0	100.00	73.9	95.49

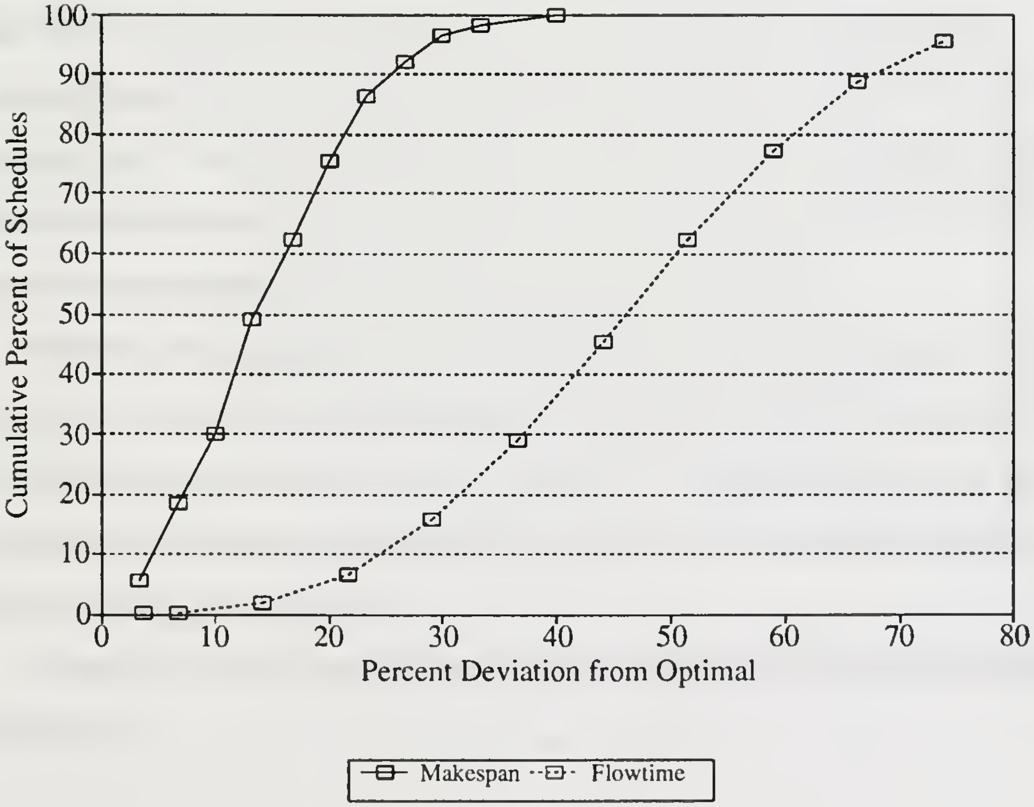


Figure 4.10. Deviation from optimal makespan and flowtime.

4.3.2 Lower Bounds

In this section we discuss lower bounds on the total flowtime. These lower bounds provide us with a way of determining the quality of the solutions produced by our heuristics and can be extended into lower bounds for a branch-and-bound algorithm.

The lower bounds are similar to those proposed in Ignall and Schrage (1965). The initial lower bound on the total flowtime is calculated by ordering the jobs by their processing times on M_0, shortest processing time first (SPT), and assuming that the second-stage machines have infinite capacity. That is, $LB_1 = \sum C_{0j} + \sum p_{1j} + \sum p_{2j}$.

We compute the second bound by considering each of the second-stage machines. Let $a = \min \{p_{0j} : J_j \in H_1\}$ and $b = \min \{p_{0j} : J_j \in H_2\}$. Now, a (b) is the earliest time that the second-stage tasks on M_1 (M_2) could start. Then, form two sequences σ_1 and σ_2 of the jobs in H_1 and H_2 respectively by ordering the jobs in each group by the second-stage task processing times, shortest processing time first. Schedule on M_1 the second-stage tasks of the jobs in H_1 in the order given by σ_1. The first task should begin at a, and each successive task should immediately follow (we do not schedule the first-stage tasks). This forms completion times C_{1j} for the jobs. Repeat for the second-stage tasks of the jobs in H_2 in order to calculate C_{2j}.

However, only one machine can start at its earliest time. The other must start at a time no less than $a + b$. Let r be the cardinality of H_1 and s the cardinality of H_2. Either the r tasks on M_1 will be delayed by b, or the s tasks on M_2 will be delayed by a. After calculating rb and sa, we can increase our lower bound by adding the smaller quantity. Thus, the second lower bound is $LB_2 = \sum C_{1j} + \sum C_{2j} + \min \{rb, sa\}$.

In Example 4.3, we calculate the lower bounds for the problem instance that we introduced in Example 4.1.

Example 4.3. Given the following five jobs in two groups:

J_j	H_i	p_{0j}	p_{ij}	t_j
J_1	H_1	2	4	9
J_2	H_1	1	1	1
J_3	H_1	2	4	5
J_4	H_2	2	1	4
J_5	H_2	1	3	3

Lower Bound 1: Order first-stage by SPT and add second-stage:

p_{0j}	1	1	2	2	2
C_{0j}	1	2	4	6	8

$\Sigma C_{0j} = 21$. $\Sigma p_{1j} = 9$. $\Sigma p_{2j} = 4$. $LB_1 = 34$.

Lower Bound 2: Order each second-stage machine by SPT:

$a = 1$. $r = 3$. $b = 1$. $s = 2$.

p_{1j}	1	4	4	p_{2j}	1	3
C_{1j}	2	6	10	C_{2j}	2	5

$\Sigma C_{1j} = 18$. $\Sigma C_{2j} = 7$. $\min\{rb, sa\} = 2$. $LB_2 = 27$.

4.3.3 Special Case

This section discusses a special case that leads to easily-found optimal solutions.

Theorem 4.7. If $p_{1k} \le p_{0k}$ for all J_k in H_1 and $p_{2j} \le p_{0j}$ for all J_j in H_2, then any sequence σ in which the first-stage tasks are in SPT order forms an optimal schedule.

Proof. Suppose we have such a σ. Consider H_1. If J_i is the first job on M_1, then $C_{1i} = C_{0i} + p_{1i}$. If J_k is the next job from H_1 on M_1, then $p_{1i} \le p_{0i} \le p_{0k}$. Thus, the second-stage task of J_i completes before (or at the same time as) the first-stage task of J_k and does not delay the second-stage task of J_k. Thus, $C_{1k} = C_{0k} + p_{1k}$. Similarly, this equality is true for all J_k in H_1, and $C_{2j} = C_{0j} + p_{2j}$ for all J_j in H_2. The total flowtime is therefore $\Sigma C_j = \Sigma C_{0j} + \Sigma p_{1j} + \Sigma p_{2j}$. Since the first-stage tasks are in SPT order, this achieves the first lower bound, and this sequence is an optimal schedule. QED.

4.3.4 Empirical Testing

Empirical testing was performed using a number of different heuristics. The three heuristics that performed the best (and very similarly) are described below. The 3MLA-MS problem sets were used as a testbed. For each problem, the four lower bounds on the flowtime were calculated. The best of these was taken as the lower bound. For the fifteen-job problems, we were able to find the optimal solutions from the branch-and-bound algorithm. The performance of a heuristic on a problem was taken as the relative deviation from the optimal solution (if known) or the best lower bound.

SPT-look-ahead. Each group H_1 and H_2 is ordered by the first-stage task processing times (shortest first), forming two sequences. These sequences are interleaved by choosing at each step the first unscheduled job from one sequence or the other. Define at each step the following variables: t_0 is the completion time of the partial schedule on M_0 and t_1 is the completion time of the partial schedule on M_1. Let p_1 (p_2) be the processing time on M_0 of the next job from H_1 (H_2). Note that these are the shortest such task processing times from each set of jobs. If $t_1 - t_0 \leq p_1$, the work-in-process inventory (WIP) waiting at M_1 is low; schedule the job from H_1. If $t_1 - t_0 \geq p_1 + p_2$, then the WIP at M_1 is high; schedule the job from H_2. Else, $p_1 < t_1 - t_0 < p_1 + p_2$, and the WIP is intermediate; schedule the job with shortest first-stage processing time (p_1 or p_2).

WIP-look-ahead. Again, order the jobs in each group by the first-stage task processing times (shortest first). Combine the groups as in SPT-Look-ahead, except for one case: If $p_1 < t_1 - t_0 < p_1 + p_2$, schedule the job from H_1. Thus the work on M_1 is always used to determine which job to schedule.

Johnson-look-ahead. Sequence each group by Johnson's rule. Combine the groups as in SPT-Look-ahead, except for one case: If $p_1 < t_1 - t_0 < p_1 + p_2$, schedule the job from H_1. Thus the work on M_1 is always used to determine which job to schedule.

We tested the heuristics on the problem sets described in Section 1.7. For the fifteen-job problems, we were able to find the optimal solutions from a branch-and-bound algorithm. We

could not find optimal solutions for the larger problems. Therefore, in order to measure the heuristics, we computed a lower bound on the flowtime for each problem. We calculated the lower bounds described in Section 2.1 and took the largest as the lower bound. The performance of a heuristic on a problem was taken as the relative deviation from the optimal solution (if known) or the best lower bound. Due to the special structure of the problem, we could find improve the second lower bound for the instances in Set 4 by determining the optimal total flowtime for the five jobs in H_1.

Minimizing the total flowtime is harder than minimizing the makespan, and in Table 4.5 we report the results of the three heuristics described above. These heuristics were selected because they performed much better than a number of other procedures that combined the groups differently. The look-ahead heuristics found solutions with average total flowtime within eight percent of the optimal value. Because of the special structure of the 3MLA instances in Problem Set LA201, very good solutions were easier to find. The WIP-Look-ahead heuristic was slightly better on all of the other problem sets. However, the heuristics were very close to each other.

Table 4.5. Heuristic performance for the 3MLA-FT Problem.
Performance relative to optimal or lower bound.

Set	SPT Look-ahead	WIP Look-ahead	Johnson Look-ahead
LA 154	7.23	7.13	7.32
LA 304	5.41	5.16	6.15
LA 504	4.95	4.85	6.58
LA 201	1.37	1.37	1.37

4.4 Minimizing the Number of Tardy Jobs

4.4.1 Problem Introduction

The other objective function under consideration in look-ahead scheduling is the number of tardy jobs. This problem models a subproblem of the shop scheduling problem related to the

feeding of a bottleneck machine. The objective mirrors the management concern of customer satisfaction.

Let us call our problem the Three-Machine Number Tardy (3MNT) problem. Recall that only permutation schedules need be considered. Thus a sequence for all of the jobs defines a unique schedule. The problem notation is the same as that for 3MLA-MS, except that we have for each job J_j a due date d_j. Since 3MLA-MS is strongly NP-complete, 3MNT is also strongly NP-complete. Consider an instance where all of the jobs have the same due date: finding a schedule with no tardy jobs is equivalent to finding a schedule with makespan less than or equal to the common due date.

Since the problem is computationally difficult, we will examine a simple lower bound, a special case, and some heuristic approaches to finding solutions.

4.4.2 Lower Bound and Special Case

Good lower bounds are hard to find for the problem of minimizing the number of tardy jobs in a flow shop (Hariri and Potts, 1989). We will make use of a fairly simple one.

The lower bound makes use of the fact that the Moore-Hodgson algorithm will find the optimal number of tardy jobs for a one-machine problem. Given an instance of 3MNT, the due date of each job is adjusted by subtracting the processing time of the second-stage task. These adjusted due dates and the first-stage processing times form a one-machine problem for machine zero. The lower bound is calculated by using the Moore-Hodgson algorithm to optimally sequence these tasks. The number of tardy first-stage tasks is a lower bound on the minimum number of tardy jobs for the 3MNT instance.

This lower bound is achievable in the following special case:

Theorem 4.8. If min $\{p_{0j} : J_j \in H_1\} \geq$ max $\{p_{1j} : J_j \in H_1\}$ and min $\{p_{0j} : J_j \in H_2\} \geq$ max $\{p_{2j} : J_j \in H_2\}$, then an optimal sequence can be found be minimizing the number of first-stage tasks that are tardy to the adjusted due dates.

Proof. Because of the conditions above, the completion time of a job J_j in H_i is $C_j = C_{0j} + p_{ij}$. Thus, J_j is tardy if and only if $C_{0j} > d_j - p_{ij}$. QED.

We also tried lower bounds on each of the second-stage machines, but these were not as good. This seems reasonable if the interaction of the two flows is significantly contributing to tardiness.

4.4.3 Heuristics

Simple rules that can be extended for this problem include the Moore-Hodgson algorithm and Earliest Due Date (EDD). These can be expanded by including look-ahead ideas. We also developed a simple problem space genetic algorithm to find good solutions.

Look-ahead rules. The look-ahead extension of the Moore-Hodgson algorithm includes both machines. The jobs are ordered by their due dates and added to the schedule until a tardy job is found. The procedure then determines the critical path of tasks that determines the completion time of the tardy job. Then, each job in the path is evaluated to determine how much the completion time would decrease if that job were removed from the partial schedule. This calculation depends upon whether the job precedes, is, or follows the crossover job (the job whose first and second tasks are in the critical path).

We developed two look-ahead versions of the EDD rule. The first is similar to the Moore-Hodgson rule. The jobs are sequenced by their due dates. They are scheduled one at a time. When a job is tardy, however, we simply remove it to the end of the schedule. (In the Moore-Hodgson rule, we look for the job whose removal helps the most.) This form of the EDD is called EDD-No Tardy.

The other look-ahead version (EDD-Look-ahead) tries to schedule machine one (the bottleneck) carefully. The jobs in each group are ordered by their due dates. The primary idea is to get the jobs from group one to be on-time; group two jobs can be inserted if they don't interfere. At any point in constructing the schedule, we consider the next job from each group. If either would be tardy if scheduled next, we place it at the end of the schedule. Eventually, we get

a job from each group that would be on-time if scheduled next. We then determine if scheduling the group two job next would cause the group one job to be tardy. If not, we schedule this group two job and consider the next job from that group. Else we schedule the group one job next.

Genetic algorithm. After initial testing, we determined that the look-ahead Moore-Hodgson rule found consistently good solutions. Thus, we used a problem space genetic algorithm to adjust the problem data used by this rule so that we could find better solutions. The procedure is similar to those described in Chapter 3. We adjust the due dates using a steady-state genetic algorithm. The population size was 50, and the algorithm generated 1000 new individuals.

4.4.4 Results

In order to compare the heuristics we created a number of problem sets. Each set had 10 similar instances. Problems were created with 15, 30, and 50 jobs. The processing times were chosen randomly, and the due dates were chosen from a range that depended upon the sum of the processing times.

The look-ahead Moore-Hodgson heuristic performs better than any of the other rules (See Table 4.6). A number of rules that used the first-stage due date of a lot were also tested but did not perform as well. The genetic algorithm was able to find slightly better solutions than those found by the Moore-Hodgson rule.

Table 4.6. Summary table of results for 3MNT.

Set	Jobs	Lower Bound	EDD No Tardy	EDD	Moore Look-ahead	Genetic Algorithm
NT 151	15	5.8	7.8	7.3	7.0	6.5
NT 152	15	5.3	7.6	7.1	7.1	6.3
NT 301	30	9.0	12.7	12.5	12.0	10.6
NT 501	50	17.0	21.6	21.4	20.7	20.4

4.5 Application to Job Shop Scheduling

One of the primary motivations for studying these problems was to see if we could develop useful dispatching rules. After considering our results, it seemed clear that look-ahead and look-behind policies similar to those we used on the one-machine and three-machine problems would be intuitively good ways to dispatch jobs. However, we wanted to determine the tradeoffs of using such rules on a number of objectives. To this end, we created a job shop scheduling problem that modeled the semiconductor test area we were studying. This problem had 82 jobs and 23 machines and included various test operations. Processing time data were gathered from some historical lots. We scheduled the jobs under a number of dispatching rule combinations, using standard dispatching rules, look-ahead rules, and look-behind rules. We measured the schedules on four scales: total flowtime, makespan, number of tardy jobs, and total tardiness.

The bottleneck in this problem was a set of burn-in boards. Thus, we developed a look-ahead rule that orders the jobs waiting by their task processing times and uses information about the downstream bottleneck resource (the burn-in board availability) to determine which lot should be scheduled next. We also developed a look-behind rule that sequences lots by EDD and reserves burn-in boards for the next late lot that will be arriving soon. The effort of using these rules is slight for our problem; in a manufacturing environment, the dispatching effort might be more significant.

We used the following scheme in order to see how look-ahead and look-behind rules would influence schedule performance. We allocated the rules by dividing the machines into three areas: electrical test, burn-in, and other. In any given policy, all of the machines in the same area used the same dispatching rule. Our model included one burn-in workstation and 15 testers. We used seven standard rules: SPT, EDD, Slack per Remaining Operation, LPT, Modified Due Date, Earliest Finish Time, and First-In-First-Out in all areas.

For each of the standard rules, we created four policies. In the first, all of the areas used that rule. In the second, the testers used the look-ahead rule (since these were the machines

feeding the burn-in area). In the third, the burn-in area used the look-behind rule. In the fourth, the testers used the look-ahead rule while the burn-in area used the look-behind rule. This yielded 28 policies.

The average results over the seven standard rules are summarized in Table 4.7. We observe that the use of a look-behind rule, which is concerned with expediting late jobs, has a drastic effect on due date-related measures. It is able to reduce total tardiness while increasing the number of tardy jobs. This is a common tradeoff in scheduling problems. The look-ahead rule, which is concerned with avoiding unnecessary delays, reduces the total flowtime and makespan objectives. Our results are the consequence of the specific definitions of these look-ahead and look-behind rules. For other problems, alternative definitions may yield different results. While our results are not proof that look-ahead and look-behind rules are the answer to solving the job shop scheduling problem, the decreases in total flowtime (when the look-ahead rule was used) and total tardiness (with the look-behind rule) did encourage us to use them in our procedures to find good solutions (discussed in Chapter 5).

Table 4.7. Performance of 28 dispatching rule combinations.

	Flowtime	Makespan	Tardy	Tardiness
All policies[a]	2,179,344	88,677	17.1	192,394
Single rules[b]	2,168,014	89,324	16.6	249,968
With Look-ahead[b]	2,135,800	88,850	16.6	243,253
With Look-behind[b]	2,222,639	88,534	17.6	138,130
With both[b]	2,190,921	88,001	17.6	138,226

Notes: a: Average over all 28 policies.
b: Average over seven policies, one for each standard rule.

4.6 Chapter Summary

In this chapter we have examined a special case of the general three-machine flow shop. In this problem, the jobs to be scheduled form two classes with different groups, and we wish to minimize the makespan, the total flowtime, or the number of tardy jobs.

We proved that minimizing the makespan is a strongly NP-complete problem, and we also identified some properties of optimal solutions and some special cases that can be solved in polynomial time. We developed an approximation algorithm, Johnson Interleaved, that can find near-optimal solutions by looking ahead to the future workload of the second-stage machines. We showed that the worst-case error bound for this procedure is one-half and that this bound is tight in the limit. We also developed a branch-and-bound algorithm that can find exact solutions to the problem.

Then we described the problem of minimizing the total flowtime. We presented lower bounds, optimality conditions, and the results of testing on selected problem instances a number of heuristics that look ahead to current workload at the second-stage machines.

Finally we examined the problem of minimizing the number of tardy jobs. We discussed a simple lower bound, a special case that achieves this bound, and a number of simple and look-ahead heuristics. We also showed that a problem space genetic algorithm can find better solutions.

These results have two contributions. First is the analysis of these three-machine problems, problems previously unstudied in the literature. We conclude from the results of our empirical testing that look-ahead heuristics can find good solutions for the problems of minimizing total flowtime and minimizing the number of tardy jobs, and the interleaving procedure minimizes makespan.

Second, while these problems are important questions in their own right, they are also significant as subproblems in a job shop. It is possible to apply our results to the problem of job shop scheduling, either as part of a general scheduling procedure or as an attempt to schedule the bottleneck of a job shop more efficiently.

CHAPTER 5
GLOBAL JOB SHOP SCHEDULING

In this chapter we describe a global job shop scheduling procedure that uses a genetic algorithm to find a good schedule. We have implemented the scheduling system in a semiconductor test area. The test area is a job shop and has sequence-dependent setup times at some operations. The concern of management is to meet their customer due dates and to increase throughput. This requires the coordination of many resources, a task beyond the ability of simple dispatching rules. We discuss a centralized procedure that can find a good schedule through the use of a detailed scheduling model and a genetic algorithm that searches over combinations of dispatching rules. We discuss our effort in developing a system that models the shop, creates schedules for the test area personnel, and contributes to test area management.

5.1 Introduction

In many areas of manufacturing, the ability of a facility to meet its objectives depends upon the close coordination of resources. This coordination must occur on different levels: capacity planning, release planning, and lot dispatching. Effective approaches exist (and new techniques are being developed) for the first two levels. The third level, meanwhile, continues to pose very difficult scheduling problems. These are the problems that we address. Like other researchers, we are interested in creating systems to better schedule resources in a manufacturing process, since effective scheduling can lead to improvements in throughput, customer satisfaction (measured by meeting due dates), and other performance measures.

We are concerned with the effective scheduling of a semiconductor test area, a job shop environment. In this facility, a lot is a number of identical semiconductor devices. Each lot must undergo a number of tests (electrical and physical) and other operations before the product can be

shipped to the customer. Associated with the lot is the due date of the customer order it will be used to fill. Because each product has a unique route, different lots take different paths through the test area.

A number of characteristics make the semiconductor test area difficult to control. These include the conflicting goals of management (balancing increased throughput against meeting due dates), work centers with sequence-dependent setup times, operations where multiple lots can be processed simultaneously, and the relationships between operations.

The purpose of this chapter is to describe the global job shop scheduling system developed for semiconductor testing. We will examine the needs of the semiconductor test area, their requirements for a scheduling system, and our approach to this problem.

The primary goal of the global job shop scheduling system is to use information about the current status of the shop, the jobs to be manufactured, and the production process in order to create a schedule of activities for each work center in the shop over a fixed time period. By using a centralized procedure with global information, the system can search for a schedule better than that computed by making nearsighted, local decisions.

The system finds good solutions with a genetic algorithm, a type of heuristic search. One interesting characteristic of this search is that it looks for a good combination of dispatching rules to find an efficient schedule. The use of dispatching rules is an effective local procedure to create a job shop schedule; such techniques do not, however, use any global information. Our procedure addresses this shortcoming. It goes beyond simply finding good local schedules; it executes a search that looks for a better global schedule. While global scheduling techniques have been proposed before, the important contributions of our work are the development of a genetic algorithm for job shop scheduling and the implementation of a scheduling system that uses such an advanced solution procedure.

The next section of this chapter will review some background and related research on job shop scheduling. The genetic algorithm for global scheduling is introduced in Section 5.3, where we also discuss a small example of our search space. In Section 5.4 we describe the scheduling

problem in the semiconductor test area under consideration and the global job shop scheduling system that we developed. We summarize our report in Section 5.5 and describe how a similar system may be useful in other manufacturing environments.

5.2 Job Shop Scheduling

As mentioned in the introduction, the problem of coordinating resources to ensure efficient manufacturing is a difficult problem. When the production process is fairly straightforward, techniques such as Just-In-Time (JIT) manufacturing result in efficient scheduling. In many situations, however, the process is much more complicated, and finding an optimal or near-optimal schedule is an impossible task. In this section we will review some of the approaches to job shop scheduling. More details on these papers can be found in Chapter 2.

The traditional method of controlling job shops is the use of dispatching rules. A dispatching rule is a sequencing policy that orders the jobs waiting for processing at a machine; the ordering depends upon the particular dispatching rule used. Common rules include the Shortest Processing Time (SPT) and Earliest Due Date (EDD) rules.

While such rules have been the subject of much research, standard dispatching rules have a narrow perspective on the scheduling problem, since they ignore information about other jobs and other resources in the shop. More advanced look-ahead and look-behind rules attempt to include more information, but their reach is still limited. Effective job shop scheduling depends upon the interaction of a number of factors. The complexity of these interactions makes the development of a global scheduling system a difficult task.

A number of researchers have looked at semiconductor manufacturing at all levels, from production planning to scheduling. Approaches to shop floor control include lot release policies, dispatching rules, deterministic scheduling, control-theoretic approaches, knowledge-based approaches, and simulation. Uzsoy *et al.* (1992a, 1993) review a substantial number of papers that consider these approaches to semiconductor scheduling.

This work in semiconductor manufacturing includes corporate-wide production planning that uses a rate-based model of production and linear programming (Leachman, 1993, and Hackman and Leachman, 1989). Such a global planning system has been developed under the name IMPReSS at Harris Semiconductor. Hung and Leachman (1992) have developed an iterative method that uses a linear program and a discrete-event simulation to develop long-term production plans for a wafer fab.

While most of the research in semiconductor scheduling has concentrated on the fabrication of semiconductor wafers, a number of other papers have addressed the problems of semiconductor test. These include Lee, Uzsoy, and Martin-Vega (1992), and Uzsoy *et al.* (1991a, 1991b, and 1992b). Lee *et al.* (1993) report on the implementation of a decision support system for the dispatching of lots in a semiconductor test area. Our work is the natural extension of this system.

Previous approaches to the production of detailed shop schedules for planning and shop floor control include expert systems like ISIS (Fox and Smith, 1984), OPIS (Smith, Fox, and Ow, 1986, and Ow and Smith, 1988), MICRO-BOSS (Sadeh, 1991), and OPAL (Bensana, Bel, and Dubois, 1988); cost-based procedures such as OPT (Optimized Production Technology, reviewed by a number of authors, including Jacobs, 1984), Faaland and Schmitt (1993), and the bottleneck dynamics of SCHED-STAR (Morton *et al.*, 1988); simulation (Leachman and Sohoni, Najmi and Lozinski); and leitstands (Adelsberger and Kanet, 1991). Adler *et al.* (1993) describe the implementation of a bottleneck-based scheduling support system for a paper bag production flexible flow shop. While these approaches have been developed for a number of different manufacturing processes, the complications of the semiconductor test area forced us to consider a new design.

Job shop scheduling, as one of the most difficult scheduling problems, has attracted a lot of attention from researchers. Techniques such as the shifting bottleneck algorithm (Adams, Balas, and Zawack, 1988) or bottleneck dynamics (see Morton, 1992, for example) concentrate on solving the problem at one machine at a time. More work has gone into the development and

evaluation of various dispatching rules. Panwalkar and Iskander (1977) present a list of over 100 rules. Recent studies include Fry, Philipoom, and Blackstone (1988), Vepsalainen and Morton (1988), and Bhasakaran and Pinedo (1991).

More sophisticated *look-ahead* and *look-behind* rules have also been discussed. Look-behind rules (called *x-dispatch* by Morton, 1992) consider the jobs that will be arriving soon from upstream machines. Look-ahead rules consider information about the downstream machines. This includes the work-in-next-queue and the number-in-next-queue rules of Panwalkar and Iskander (1977), bottleneck starvation avoidance (Glassey and Petrakian, 1989), and lot release policies that look-ahead to the bottleneck (Wein, 1988; Glassey and Resende, 1988; and Leachman, Solorzano, and Glassey, 1988). Look-ahead and look-behind scheduling problems have been studied in this dissertation and by Lee and Herrmann (1993).

Finally, heuristic searches have also been developed for job shop scheduling, and a number of these are discussed in Section 2.8 of this dissertation.

5.3 A Genetic Algorithm for Job Shop Scheduling

In this section we will describe how a genetic algorithm can be used to find good schedules for the job shop scheduling problem.

One approach to difficult scheduling problems such as job shop scheduling is local search, an iterative procedure that moves from solution to solution until it finds a local optimum. *Smart-and-lucky searches* (or heuristic searches, or probabilistic search heuristics) attempt to overcome the primary problem of these simple searches: convergence to local optima. These more complex searches are smart enough to escape from most local optima; they still must be lucky, however, in order to find the global optimum.

In previous sections we reviewed the basic concepts of genetic algorithms and mention some application of smart-and-lucky techniques for job shop scheduling. In this section we describe our application of these ideas.

5.3.1 The Heuristic Space

Many heuristic searches for the job shop scheduling problem have been considered. One recently-introduced idea is to search the problem and heuristic spaces, since a solution is the result of applying a heuristic to a problem. A change to the parameters of a heuristic or a problem yields a slightly different solution. This section will describe how a heuristic space can be used for job shop scheduling. We will show why a genetic algorithm is especially suited for searching this space.

The idea of searching heuristic and problem spaces was reported by Storer, Wu, and Vaccari (1992). In their paper, the authors examine the general job shop scheduling problem. They define a heuristic space composed of vectors of dispatching rules. Each vector in the space can be used to determine a schedule. Each rule in the vector is used for a fixed number of dispatching decisions, regardless of the machine being scheduled. That is, all of the machines use the same dispatching rule at the same time until the next rule replaces it.

Our approach uses a different perspective. Since we will be working in a dynamic job shop scheduling environment, we may not know how many operations will be scheduled. Thus, it is impossible to divide the scheduling horizon by allocating each rules to a fixed number of operations. Instead, it seems more fair to assign a dispatching rule to each machine. The system can evaluate this combination of dispatching rules (which we call a *policy*) by measuring the performance of the schedule that is created by using this policy. Changing the policy by modifying the dispatching rule on one or more machines changes the schedule created.

In order to demonstrate this idea, consider the following four-job, three-machine problem and two policies used to dispatch the jobs waiting for processing (see Figure 5.1 for the task processing times).

Job	Due Date	Sequence of Machines to Visit:
J_1	8	Machine 1, Machine 2, Machine 3
J_2	12	Machine 1, Machine 2
J_3	17	Machine 2, Machine 1, Machine 3
J_4	15	Machine 2, Machine 3

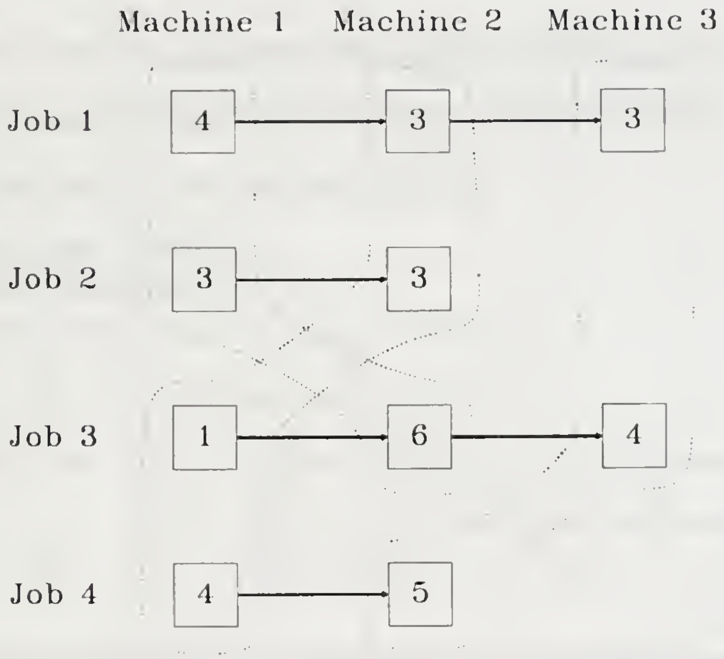
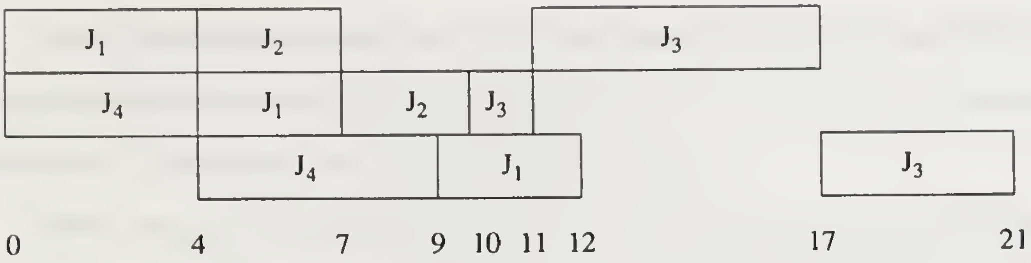


Figure 5.1. Task processing times for each job.
Tasks grouped by a dotted line are on the same machine.

Policy 1 = [EDD, EDD, EDD]. Under this policy, the tasks waiting for processing at M_1 are sequenced by earliest due date first. Thus, when both J_1 and J_2 are at M_1, J_1 is sequenced first since its due date is 8.

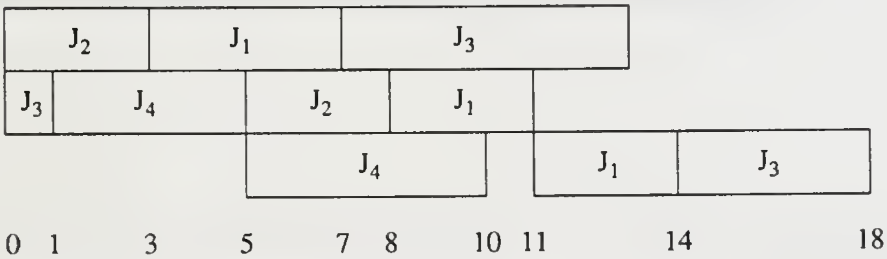
The dispatching rule for M_2 is EDD, so the tasks are sequenced by the job due date. When J_3 and J_4 are at M_2, J_4 is selected. After this task is completed, J_1 has arrived and its due date is also smaller than J_3. The same thing occurs when J_1 finishes and J_2 simultaneously arrives.

Policy 1 : [EDD, EDD, EDD]
Schedule: Makespan = 21.



In Policy 2, the altered dispatching rules, [SPT, SPT, SPT], change the selection of jobs. On M_1 the job J_2 has the shortest task processing time and is thus preferred. On M_2 the job J_3 has the shorter task processing time and is scheduled first.

Policy 2 : [SPT, SPT, SPT]
Schedule: Makespan = 18.



Obviously, the application of a different policy creates a different schedule with a different evaluation on any objective we could wish to measure (makespan, defined as the maximum job completion time, was chosen here only as an example).

These policies can be easily manipulated by a genetic algorithm. Traditional genetic algorithms must be modified for machine scheduling problems since the components of the strings (the jobs in the sequence) are not independent of each other. The heuristic space described above, however, consists of vectors that have independent elements. That is, the dispatching rule for the first machine does not affect what values the other elements can have. Thus, a crossover operation that breaks two strings (vectors) and joins the separate pieces yields offspring that are valid points in the search space.

5.3.2 A Genetic Algorithm for Global Scheduling

We will describe in this section the Genetic Algorithm for Global Scheduling (GAGS). This search procedure is the engine that finds a good schedule. After discussing it in this section, we will turn away and address the scheduling system that it drives.

GAGS consists of two primary components: the genetic search and the model of the shop floor. The interaction between these two functions is outlined in Figure 5.2. The genetic algorithm starts with a number of policies. (Each policy is a combination of dispatching rules, one rule for each machine. See Table 5.1 for a list of the dispatching rules that were used in the global scheduling procedure.) Each policy is evaluated by the schedule that is created if the jobs in the shop are dispatched according to this policy. The model of the shop floor is employed to build this schedule from the set of shop, job, and process information.

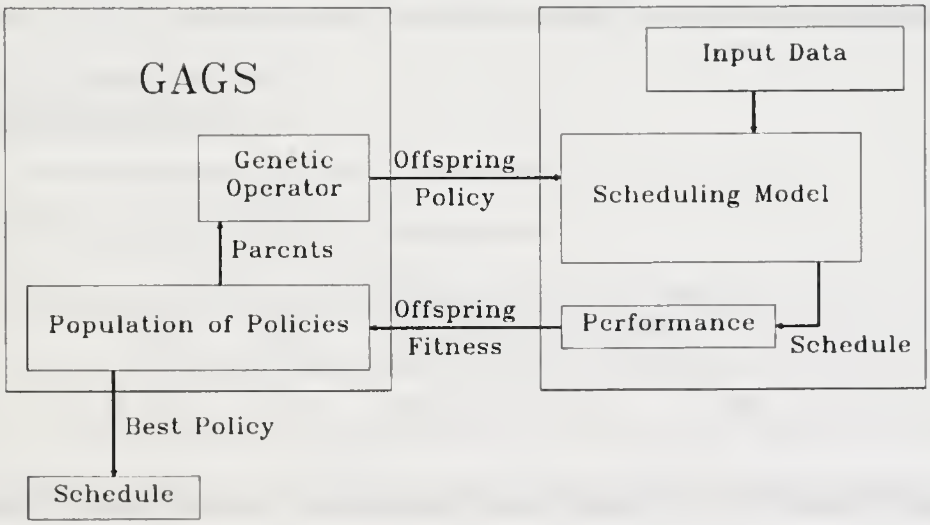


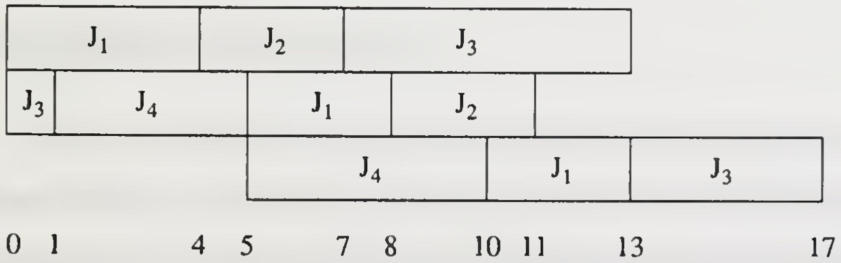
Figure 5.2. GAGS - Scheduling Model Interface

Table 5.1. List of dispatching rules considered.

SPT
Minimum Setup
EDD
Shop: Late-Setup Match-EDD
Slack per remaining operation
Modified Due Date
Longest Processing Time
Shortest Remaining Processing Time
Earliest Finish Time
First In First Out
Least work in next queue
EDD Look-ahead to burn-in
SPT Look-ahead to burn-in
Look-behind from burn-in

The genetic algorithm creates new offspring policies by combining two policies in the population (crossover) or changing a solitary parent (mutation). For instance, consider again the above three-machine problem and the two policies [EDD, EDD, EDD] and [SPT, SPT, SPT]. These policies were evaluated by creating schedules for all three machines. Their evaluations (on the makespan objective function) were 21 and 18. If we split each policy after the first rule and link the beginning of the first policy with the end of the second policy, we create the offspring policy [EDD, SPT, SPT], which the reader can confirm creates a schedule with a makespan of 17.

Policy 3 : [EDD, SPT, SPT]
Schedule: Makespan = 17.



These offspring policies are evaluated using the scheduling model. The genetic algorithm converges to a number of good policies, and the best is used to create a final schedule.

The model of the shop floor is the problem to be solved, and the genetic algorithm provides different heuristics. The system depends upon this model, therefore: only with a valid model can

a scheduling system create plans that match reality. For our project, the model was a deterministic simulation of the shop. It uses as input a set of resources that correspond to the equipment and staffing of the test area, a set of jobs that correspond to the lots that need processing in the test area, and expert knowledge about the production processes. The policy provided by the genetic algorithm sequences the jobs at each resource. Events in the simulation correspond to resources beginning work on a task and resources finishing tasks. Other events are added as necessary to control the simulation of the production process.

Note that the search is not affected by the complexity of the scheduling problem being solved. The genetic algorithm can find solutions to classical problems and to problems with previously-unconsidered characteristics. The genetic algorithm can take advantage of complexity by including dispatching rules designed for use in that environment.

5.4 Global Job Shop Scheduling

In this section we will discuss the manufacturing process and scheduling needs of the semiconductor test area for which we are trying to create good schedules. As we will see, the problem is quite difficult. This will lead us into the description of the design, implementation, and contributions of our global job shop scheduling system.

5.4.1 The Semiconductor Test Process

The manufacturing of semiconductors consists of many complex steps. This includes four primary activities: wafer fabrication, probe, assembly, and test. This research is concerned with the last facility. Although the routes of lots through the test process vary significantly over the many different types of products, general trends can be described (see Section 2.1 of this dissertation).

The area under study tests commercial-use semiconductor devices and consists of two domains: one dedicated to Digital products and another to Analog products. We directed our work toward the Digital side, where there are over 1400 product lines. The test area has three

shifts per day, five days a week. The resources in the Digital side include nearly sixteen electrical test heads and a dozen branders. The staffing on a normal shift consisted of nearly fifteen personnel in eight areas. The product mix changes continuously, and staffing and machine resources often change to reflect this.

Although a typical product may have a route that consists of over 30 numbered operations, only a fraction of those are steps where significant time and resources are required. In our job shop model of the test area, we concentrated on the following operations: electrical test (room, low, and high temperature), brand, burn-in, burn-in load and unload, visual-mechanical test, and document review. Due to the re-entrant nature of the process, an average lot will have twelve steps that need processing.

Most of the operations in the semiconductor test area are electrical or physical tests, performed in an automated fashion or by hand. During these tests, each device in a lot must be examined. Thus, the processing times depend directly upon the size of the lot. Also, the products in the test area have different package types. A package is the shell in which the semiconductor chip resides. These packages can be plastic or ceramic and come in many different sizes and styles. The package of a product also influences how the devices are tested.

In the job shop model, a work center for each of the resources in the shop (except the burn-in ovens) is a tester, machine, or operator that needs to be scheduled. Each type of work center has a list of the operations that can be performed at the station. Distinct resources that perform the same operations are modelled as separate stations, each with a queue of jobs that next need processing at that station. These queues are sequenced by the dispatching rule for that station.

Two unique operations in semiconductor testing are electrical test and burn-in. Electrical test operations have sequence-dependent setups, and burn-in is a bulk-service process.

The sequence-dependent setups at electrical test are a result of the test requirements. First, a handler must be attached to the test head to automatically feed the devices in the lot. However, due to the physical attributes of the packages, different handlers are required for different packages. Testing also occurs at various temperatures, which requires additional equipment. The

amount of setup is thus affected by the setup of the previous test, since it will be necessary to put into place new equipment only when the previous setup was different.

The other unique operation is burn-in. At this operation, the devices in the lot are placed into one of a number of ovens for a fixed period of time. These times range from 24 hours to over 4 days. Many lots can be burned-in at once, and the capacity constraint is more often on the availability of burn-in boards than on the space in the ovens. Additionally, after the boards are removed from the oven, the devices are unloaded from the boards, and the lot must undergo an electrical test within 96 hours to locate any faulty devices. The combination of bulk service, secondary capacity constraints, and operation deadlines requires special modeling.

Access to burn-in is controlled by the burn-in load work center, since each lot must be loaded onto burn-in boards before being placed in the oven. If sufficient boards are not available, the lot cannot be processed. After burn-in load, the lot moves directly to the burn-in oven and begins the burn-in period. The factory control system provides information on which day the lot should be removed from burn-in. On that date, the lot is scheduled for unload, and the burn-in boards become available for another lot.

As Lee *et al.* (1993) mention, the test area has five features that distinguish it from classical job shop scheduling:

1. Sequence-dependent setup times and re-entrant product flows,
2. Machines with different scheduling characteristics,
3. Complex interactions between machines,
4. Dynamic production environment, and
5. Multiple, conflicting objectives.

These characteristics make effective scheduling of the semiconductor test area a difficult task. They also encourage us to design a global scheduling system that can search for good solutions without being obstructed by complexity.

5.4.2 The Previous Scheduling System

In this subsection we briefly discuss how scheduling was done previously in the test area using dispatching rules and some drawbacks to this system. (The development of this system is described in Lee *et al.* 1993.)

The test area was using a set of dispatching stations to sequence the lots awaiting processing at each work center at the beginning of the shift. During the shift, this ordered list of lots was updated by using the dispatch station to resequence the lots in the queue, since more lots may have arrived. The dispatching stations and rules are part of a software module called Short-Interval Scheduling (SIS).

SIS is a component of WORKSTREAM, a product of Consilium, Inc. (Consilium, 1988). WORKSTREAM is the test area's computer-integrated manufacturing (CIM) system and is implemented on the corporate VAX mainframe. Transactions such as processing a lot or beginning a setup are logged into WORKSTREAM, which maintains information about the current status of each lot and each machine. This information is used by SIS to sequence the lots waiting for processing.

In addition to WORKSTREAM, the company uses higher-level systems such as Activities Planning & Dispatching (AP/D) to match the production lots to customer orders and IMPReSS to determine the amount of product that each area of the company (including the test area) should be produced each week. All of these systems provide critical data about the lots to be processed and the characteristics of the test area. We will need this information to model the test area.

While SIS has led to better scheduling in the test area, it has a number of disadvantages related to the structure and capabilities of the CIM system. The primary obstacle is that dispatching rules are making local decisions (even when they attempt to look-ahead or look-behind). Thus they are unable to know how their decisions affect the work at other areas in the test area. And they are unable to advantage of information that may lead to better dispatching

decisions. In addition, the dispatching rules are unable to use information about processing and setup times.

Since the implemented dispatching rules cannot use global information, we began to consider job shop scheduling procedures. While a number of optimization procedures have been suggested (see Section 2), these have concentrated on the classical job shop problem of minimizing makespan. In particular, the more well-known procedures use the disjunctive graph to represent the problem. Unfortunately, manufacturing environments like semiconductor test area often have more complicated problems that require more complicated scheduling models.

We wanted to make use of global information in a complex production process and to search for a better schedule. Therefore, we decided to implement the genetic algorithm for global scheduling. We described the primary characteristics of this procedure in Section 3. In the remainder of Section 4 we will describe the development of our global job shop scheduling system.

5.4.3 Scheduling Needs

Over the course of a number of years, the scheduling system in the semiconductor test area progressed from manual dispatching to an integrated rule-based decision support system. As we mentioned above, this system had significant limitations, since it was based on dispatching rules. However, the planners in the area need to know in what order the lots waiting at a station should be processed, and this system gave them this information. They would also like to know when these lots will be completed.

In addition to these tactical decisions, the planners need to have some idea of how their limited resources (in both personnel and equipment) on the shop floor affect the performance of the shop in relation to meeting the shop goals. Finally, they need to measure the performance of the personnel on the floor.

The managers of this test area have two objectives. The first is to meet customer due dates. The company stresses customer satisfaction, and the responsibility of the test area is to ship the

required number of parts to the customer by the requested time. The test area is measured on the number of delinquent line items, the number of customer orders that are not shipped on-time. Thus, the most important of the two primary objectives is to minimize the number of tardy jobs.

The second objective is to test as much product as possible. This goal is both positively and negatively correlated with the first goal. If the area can test product more quickly, it is more likely to meet customer due dates. However, the push to increase the quantity tested can interfere with the need to process lots on-time (since the lots with the earliest due dates may be those with the largest processing times). At the time we were working on this project, the test area had extra capacity, so we considered the objective of minimizing flowtime to be a subordinate one.

5.4.4 Scheduling System Design

We have looked at some of the complexity that occurs in scheduling a semiconductor test area. These characteristics make the problem in the semiconductor test area much more difficult than the job shop problems previously considered. At the same time, however, the CIM systems in place are able to provide the type of area-wide information needed to create a schedule. Therefore, we are motivated to try a new approach: a global scheduling system that uses a genetic algorithm to find good schedules in the presence of shop floor complexity. In this section we will describe the basic design features.

This system includes a simulation model of the test floor and an optimization procedure that can search for schedules using a genetic algorithm. The search finds a schedule that is better than any schedule that a predefined set of dispatching rules could create. This schedule specifies the order in which the lots should be processed and the times at which the operations should be done.

The system can be used to react to unforeseen events that might occur during the course of a shift by including new information and producing a new schedule; for example, a machine may fail, or an important lot may arrive. The system also serves in a decision support manner, allowing the planners to determine the effects that changing resources has upon the schedule.

Through the use of fair processing times in an accurate model, the schedule can be used as a target for the shift; the work completed by the shift personnel can be compared to the work on the schedule.

The foundation for the system is the test area's extensive CIM system, which will provide the information necessary to model the test area. Finally, the global scheduling system is under the control of production planners, who use the computing power of the system to evaluate alternative plans and to create a detailed schedule for the shop floor.

The most important component of a global job shop scheduling system is the model of the shop floor. Only with a valid model can a scheduling system create plans that match reality. For our project, the model was a deterministic simulation of the test area. It uses as input a set of resources that correspond to the equipment and staffing of the test area, a set of jobs that correspond to the lots that need processing in the test area, and a set of dispatching rules that sequence the jobs at the resources. Thus, the model creates a schedule for the current scenario.

This is the central relationship in the system. Because the schedule depends upon the dispatching rules, the optimization procedure in this global scheduling system is a search of the combinations of dispatching rules in order to find a good schedule (one that meets management goals efficiently). A genetic algorithm is employed to search over the rules, evaluating each set by running the simulation with those rules and measuring how well the schedule performed at keeping jobs on-time. This part of the system is called the Genetic Algorithm for Global Scheduling, and it is discussed in Section 5.3 of this chapter.

5.4.5 Information Requirements

The scheduling system makes use of data from a number sources inside and outside the test area's CIM system. In this subsection we will mention the types of information that are used in the system. See Figure 4 for a diagram of how the primary data structures interact.

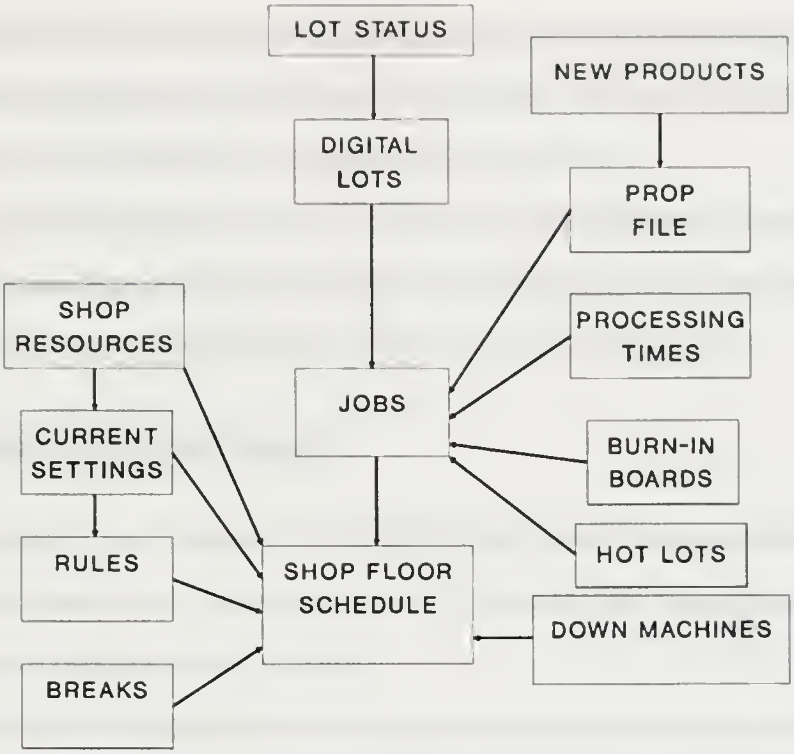


Figure 4. Data Structures

Three general types of files can be described: data extracted from the CIM, data collected by the project team, and data entered by the user. The first category includes data that is fairly stable and data that constantly changes. Stable data includes information about the product routes. This Product-Operation (PROP) file matches each product to the operations that need to be performed. This file is updated weekly as new products are added to the product mix. Other files of extracted information list package types, burn-in board requirements, and tester requirements.

The dynamic data is primarily the lot status information. The status of a lot changes as it undergoes different operations. Before each shift the CIM system takes a snapshot of the status for each lot in the test area. Our system extracts lot status information for each digital lot from this report. Although the status of a lot continues to change during the day, our system cannot see the changes until the next report.

The project team also collected data on processing times and test area resources. The collection of processing times will be discussed in Section 4.7. The resources files matches each type of resource (tester, brander) to the operations that it can process.

Finally, the user controls other information needed for the scheduling system, including the current test area resources (how many of each type), the dispatching rules to use, the breaks schedule, the status of any down machines, and the arrival of any new hot lots.

5.4.6 Implementation of Global Scheduling

The implementation of the global scheduling system followed the design of the system and the collection of data sources. In this subsection we will describe how we implemented the system and how the test area creates schedules.

After creating the scheduling model and the genetic algorithm off-line, we installed the basic programs on the corporate mainframe. We began testing the system and developing the utilities that collect the necessary job, shop, and process information. Working closely with the test area personnel, we began to run schedules each day. Feedback from these initial schedules led to improvements in all parts of the system. A technology transfer session formally introduced the system capabilities to the test area personnel, and soon they began to run the system themselves. The project team created user interface programs and documentation so that the test area would be able to use, understand, and maintain the system.

The scheduler performs shift scheduling for the Digital side of the test area. Output from the system includes the shift schedule, which lists each work center, the lots to be scheduled, and the approximate start and finish times for each operation. Also available are post-shift performance reports that compare the production of the test area to the schedule.

User interface programs make it easy for the users to collect and modify data, create a schedule, view the output, and create performance reports.

Creating a schedule consists of a number of steps. First, the current status of all of the lots in the test area is determined from the pre-shift lot status report. Non-digital lots and lots in a stores operation are ignored.

Second, the lot status information is combined with the PROP file, which converts lots into jobs. The job consists of relevant lot information and a number of operations. The lot information includes product class, due date, and lot quantity. For each operation a processing time is computed along with a remaining cycle time.

Third, the user must check the current shop settings on resources, dispatching rules, and breaks. The user is shown the default settings and has the opportunity to make any changes.

Fourth, the schedule is created using the job, shop, and process information collected to this point. This schedule can be created using the genetic algorithm, which searches over combinations of dispatching rules, or the dispatching rules which the user chose. In the latter case, no search is performed; the scheduling model uses the user's rules and outputs the schedule that these rules yield.

The genetic algorithm begins with a population of randomly-created policies. The genetic operators shift the population towards more successful policies (measured on their ability to create a schedule with more on-time jobs). The algorithm stops after a fixed period of time; this time was selected after experimentation into the trade-off between schedule quality and search effort.

In addition to creating a shift schedule, the system creates a number of other files that are used to drive performance reports. There are three performance reports: shift summary, daily summary, and daily detailed. The summary reports compare what was done with what was scheduled. The scheduled operations for each machine are compared to the operations that have been those lots. The detailed report compares processing times. The scheduled processing times are compared to the actual times that are derived from daily lot history files.

If the user makes no changes to the default shop settings, the entire process can take as little as ten minutes. This compares to the 15 minutes necessary in SIS to sequence just the lots

waiting at one work center. Therefore, the process can be done as part of the pre-shift planning, and the user has ample opportunity to update any data and to experiment with different shop settings.

The system is currently being used by the test area personnel, who will soon be expanding the system to include the Analog side of the floor. After discussing some of the issues that our implementation raised, we will describe the contributions of our system.

5.4.7 Implementation Issues

Researchers engaged in projects like this one often encounter difficulties that are considered in no classroom and occasionally learn things that are in no textbook.

Processing times. One primary problem was a result of our attempt to model an uncertain process with a deterministic procedure. We wanted to use processing times that were realistic but that also set fair targets for the test floor. Thus, we needed good estimates of what the processing times should be. The collection of these estimates was an important concern. We had historical data from the factory control system, which monitors when each lot begins and ends each operation. We began with an average for each operation, but this average ignored the variability in processing time due to lot quantity. We then took the historical data for the subset of products and fitted a linear regression to the data in an effort to predict processing time from lot size. The regression was run for different package types in order to remove another source of variability.

Perfection. We learned that perfection is an unattainable goal in a setting as complex as semiconductor test. However, we also learned that falling short of perfection is sufficient if our system improves the ability of the facility to satisfy customer demand efficiently. From the beginning, management knew (better than the project team did) that we could not hope to capture all of the activities that occur. This attitude allowed us to build a significant model instead of being overwhelmed by the complexity of finding an optimal solution. With modeling and controlling the test area perfectly beyond our reach, we concentrated on developing a system

which meets the needs of the test area planners and provides them with a tool which they can use to intelligently manage their facility.

Simplicity. A significant feature of the implemented system is the ability of production planners to modify data related to resources and other factors. This called for a number of modules that could manipulate the data without requiring undue effort from the planners. These modules were user-friendly (easy to understand and fail-safe), flexible routines that the planners felt comfortable using. We feel that this has contributed to the success of the implementation.

Timeliness. Next to the estimates of processing times, accurate information about lot status was the hardest data to gather. Since there are hundreds of lots in the facility's inventory, determining which lots are waiting at which stations is not a trivial task. The system depends upon WIP extracts that were run from the factory control system before each shift. Although the planners have the ability through the factory control system to check on the status of individual lots, it was not possible to develop procedures to gather the status of the lots during the middle of the shift.

Because the system could not gather the current lot status during the shift, it could only react to unforeseen events by rebuilding the original schedule from the beginning of the shift and incorporating the new information about machine breakdowns and lot arrival. Searches to find good schedules were not possible.

Optimization procedure. A number of other researchers continue to work on ways to solve the job shop scheduling problem. Although we cannot guarantee that our heuristic space genetic algorithm is the best procedure for finding a good shop schedule, a search over dispatching rules seemed to be an ideal approach for the needs and requirements that we faced, however.

The system makes use of the scheduling model and a separate genetic algorithm program. The two procedures are distinct modules developed and modified independently. This gave us a great deal of flexibility as we began to test and implement the scheduling system. We chose the genetic algorithm to gain the power of its parallelism and its simplicity and to avoid the problems

of local searches. The test area personnel could understand how policies combined to form new policies and how these policies could be used to create schedules.

Due to the impossibility of finding an optimal schedule to the dynamic job shop scheduling problem, it is sufficient for our heuristic space search (an approximation algorithm) to determine a quality schedule quickly.

5.4.8 Contributions of Global Scheduling

In this section we will discuss the benefits of our implementation of the global scheduling system. While primarily unquantifiable, the gains are real.

The test area now has a tool that can substantially improve scheduling with a combination of global information, detailed scheduling model, genetic search for good schedules, reaction to unforeseen events, and short-term performance reports.

The global system has a number of strengths compared to SIS, the previous scheduling system. As a centralized procedure, it can make use of information from around the test area including processing times, queue lengths, current setups, resource availability, and job arrivals. The genetic algorithm searches for a schedule that has more on-time lots than a schedule created by any fixed set of dispatching rules.

As a shift scheduler, the system gives the test area a plan that shows how work at one area depends upon work at another and that can be used to measure the performance of that shift. In fact, the ability to accurately model the test area and compute a shift schedule was just as important to the test area as the ability to find better schedules. The simulation component of the system allows the test area planners to forecast how modifying the level of available resources will affect the output of the test area. The system can also react to certain unforeseen events that may necessitate a change in the schedule.

As mentioned earlier, the product mix in the semiconductor test area is constantly changing. This variety is a significant factor on the test area performance. Thus, we are unable to measure any long-term quantitative benefits. The test area managers have expressed their

confidence that the system will lead to improvements in the performance of the test area through improved schedules and planning tools.

5.5 Chapter Summary

In this chapter we have described the development of a global scheduling system that uses a genetic algorithm to find good schedules. This system has been successfully implemented in a semiconductor test area. Controlling the test area is a complex, dynamic job shop scheduling problem where it is difficult to meet the management objectives of satisfying customer demand on-time and increasing throughput. Our scheduling system uses the extensive data available in the CIM databases in order to simulate the operation of the test area. This system uses a genetic algorithm to search for combinations of dispatching rules that yield schedules that are better than the sequencing that could be done with fixed dispatching rules. The primary practical accomplishment of this research is the implementation of an advanced job shop scheduling system in a manufacturing environment. Moreover, this implementation makes use of a new heuristic procedure that searches the combinations of dispatching rules to find a good schedule. Thus it is able to adapt each shift to changing conditions in the jobs to be scheduled and the shop resources.

This approach could be extended to any manufacturing area that has a complicated shop scheduling problem and a computer-integrated factory control system that can supply the necessary data for such a global scheduling system. In fact, the semiconductor manufacturing firm where we have implemented the system is considering exporting this system to other areas. Additionally, other optimization techniques may be useful in finding good schedules.

CHAPTER 6
SUMMARY AND CONCLUSIONS

This dissertation has reported on a number of production scheduling problems that were motivated by considering the testing of semiconductors. The research into these topics, summarized below, adds to the body of knowledge about scheduling. This work is especially relevant to the study of the harder and broader problem of job shop scheduling. Benefits of the work include results on specific one-machine class scheduling problems, results on three-machine look-ahead problems, and the use of genetic algorithms and new search spaces on different types of scheduling problems.

6.1 One-machine Class Scheduling Problems

The research on the three one-machine class scheduling problems has yielded a number of results. Most notably, the problem space genetic algorithm is a robust tool for finding high-quality solutions to difficult scheduling problems.

For the problem of minimizing total flowtime subject to deadline constraints (CFTS), we developed an multiple-pass heuristic that makes use of an optimal property for jobs in the same class. By considering the effect of wasted setup time, it is able to find reasonable solutions. We can improve upon these solutions with a problem space genetic algorithm that adjusts the job deadlines in order to create better schedules.

For the problem of minimizing the number of tardy jobs where the jobs have non-zero release dates (CSRDD), we extend a non-setup procedure to create a heuristic for the class scheduling problem. The average performance of the heuristic is good compared to a number of other dispatching rules. A problem space genetic algorithm is able to find better solutions on some especially difficult problems where the heuristic performs poorly.

Our extended heuristics for the problem of minimizing the total flowtime of jobs with non-zero release dates (FTSRD), were outclassed by a decomposition procedure and a problem space genetic algorithm. We developed a number of dominance properties for use in a branch-and-bound technique.

6.2 Look-ahead Scheduling

The three-machine problems that have been studied show that look-ahead rules can perform better than standard dispatching rules.

For the problem of minimizing makespan, the interleaving of the Johnson sequences is able to provide near-optimal solutions. The worst case relative error of this heuristic is fifty percent. There are, however, special cases of the problem that can be solved in polynomial time.

The problem of minimizing the total flowtime is more difficult. There do exist special cases where the lower bounds can be achieved. Look-ahead rules that consider the queue at the second-stage machines are able to find good solutions.

The last problem was that of minimizing the number of tardy jobs. Again, special cases exist where optimal solutions can be easily found. Look-ahead rules were able to find better schedules than other sequencing rules. A problem space genetic algorithm found improved solutions.

6.3 Searching for Job Shop Schedules

This research has investigated the development of a procedure for the job shop scheduling problem. The genetic algorithm makes use of known heuristics (dispatching rules) but increases their effectiveness by searching over combinations of rules to find a good schedule. This procedure provides a way to find good schedules under any objective function and in any scheduling environment, since it makes use of a detailed shop floor scheduler. These characteristics make this procedure unique.

In addition, this procedure has been implemented as part of a global scheduling system for a semiconductor test area. In this environment, it creates real-time schedules for the next shift using information about the current status of the lots, the current resources in the shop, and the manufacturing process. The system includes not only the simulation model and the genetic algorithm but also utility functions for the collection of data from multiple sources and the generation of performance reports. In addition, the system can respond to unforeseen events, and the test area planners can use the system to determine the effect of changing the shop resources.

6.4 Conclusions

The scheduling of a manufacturing process is a complicated problem. The static job shop scheduling problem is incredibly difficult to solve, and no system has been able to optimize the scheduling of an entire dynamic job shop, which is the environment present in many manufacturing facilities. Thus, continued research into procedures that can find good schedules is necessary.

Many researchers have studied this problem, introducing systems which range in scope from company-wide planning to machine scheduling. This research is concerned with efforts at the level of the shop floor and machine.

This research investigated production scheduling problems that are motivated by semiconductor test operations and are expected to hold widely in other production environments. Of particular concern are those problems that occur in the testing of semiconductor devices. Let us now take a moment to provide some perspective.

This dissertation is concerned with two types of operations research: *management science* and *management engineering* (the terms of Corbett and Van Wassenhove, 1993). Management science is the discovery of new results that add to the body of knowledge about a subject. Management engineering (a less active area of the field) is the solution of a practical problem by modifying existing tools or by using existing tools in original ways.

The scientific contributions are clear. While much research in the areas of scheduling and semiconductor manufacturing has been performed, this research investigated a number of previously unstudied problems and methods. Our research into these problems has yielded a number of useful properties and effective heuristics.

This research has shown that smart-and-lucky searches and the new problem and heuristic spaces can be used for the problems under consideration. Problem space genetic algorithms can find good solutions to machine scheduling problems. For the job shop scheduling problem, a genetic algorithm can search combinations of dispatching rules and use a shop floor simulation to determine a good schedule.

Additionally, this research ties together separate problems in an effort to improve the scheduling of the manufacturing process being studied. This is an engineering question. The cooperation of the semiconductor test facility motivated research into the problems of an actual system and provided an opportunity to implement our solution procedures. (This does not preclude the potential of our approach to solve problems in other manufacturing environments.)

Our global job shop scheduling system (with its detailed simulation model and heuristic space genetic algorithm) is a new technique for the problem of creating good shift schedules for a semiconductor test area in real-time. Since look-ahead dispatching rules can be more effective than standard rules, the results of the work into the three-machine subproblems have been used as dispatching procedures for the shop floor and as part of the job shop scheduling procedure.

This research opens some chapters of scheduling that need to further pursued. While the problem space genetic algorithm is a robust procedure that finds good solutions, better solutions to particular problems may be achievable through the use of solution-specific heuristics that can improve the schedules that the genetic algorithm constructs. Also, it may be possible to create searches that combine different spaces for other types of combinatorial problems.

The set of class scheduling problems includes a number of other interesting problems; so does the set of look-ahead problems. Even more interesting is the combination of these problems.

Look-ahead class scheduling problems may yield more insights into the job shop scheduling problem.

There also exist issues in global job shop scheduling that still need to be addressed: job release, resource planning, and uncertainty in the manufacturing process. It is difficult to obtain satisfactory solutions in these circumstances using current scheduling techniques.

The general approach of this research (problem and heuristic space searches) can be applied to almost any difficult problem. It may be profitable to consider this approach for problems where good solutions are hard to find.

Why investigate production scheduling problems motivated by semiconductor manufacturing? Because organizations and individuals often have difficulty meeting their goals efficiently, and this dissertation, which has applications beyond the problems contained herein, offers some ways to solve people's problems.

REFERENCES

Aarts, E.H.L., and P.J.M. van Laarhoven, "Statistical cooling: a general approach to combinatorial optimization problems," *Philips Journal of Research*, Vol. 40, p. 193, 1985.

Adachi, T., C.L. Moodie, and J.J. Talavage, "A pattern-recognition-based method for controlling a multi-loop production system," *International Journal of Production Research*, Vol. 26, pp. 1943-1957, 1988.

Adachi, T., C.L. Moodie, and J.J. Talavage, "A rule-based control method for a multi-loop production system," *Artificial Intelligence in Engineering*, Vol. 4, pp. 115-125, 1989.

Adams, J., E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, Vol. 34, pp. 391-401, 1988.

Adelsberger, H.H., and J.J. Kanet, "The leitstand - a new tool in computer-integrated manufacturing," *Production and Inventory Management*, Vol. 32, No. 1, pp. 43-48, 1991.

Adler, L., N. Fraiman, E. Kobacker, M. Pinedo, J.C. Plotnicoff, and T.-P. Wu, "BPSS: a scheduling support system for the packagin industry," *Operations Research*, Vol. 41, No. 4, pp. 641-648, 1993.

Ahmadi, J.H., R.H. Ahmadi, S. Dasu, and C.S. Tang, "Batching and scheduling jobs on batch and discrete processors," Anderson Graduate School of Management, UCLA, 1989.

Ahmadi, R. H., and U. Bagchi, "Improved lower bounds for minimizing the sum of completion times of *n* jobs over *m* machines in a flow shop," *European Journal of Operational Research*, Vol. 44, pp. 331-336, 1990.

Ahn, B.-H., and J.-H. Hyun, "Single facility multi-class job scheduling," *Computers and Operations Research*, Vol. 17, No. 3, pp. 265-272, 1990.

Atherton, R.W., "Factory scheduling using simulation models," Proceedings of the Third Symposium on Automated Integrated Circuit Manufacturing, Electrochemical Society, Princeton, New Jersey, 1988.

Atherton, R.W., and M.A. Pool, "Validated simulation models for factory control," International Semiconductor Manufacturing Science Symposium, Semicon West, Burlingame, California, May 22-24, 1989.

Bagchi, U., and R.H. Ahmadi, "An improved lower bound for minimizing weighted completion times with deadlines," *Operations Research*, Vol. 35, pp. 311-313, 1987.

Bai, X., and S.B. Gershwin "A manufacturing scheduler's perspective on semiconductor fabrication," VLSI Memo No. 89-518, Massachusetts Institute of Technology, 1989.

Bai, S.X., and S.B. Gershwin, "Scheduling manufacturing systems with work-in-process inventory," Proceedings, Twenty-ninth IEEE Conference on Decision and Control, Honolulu, Hawaii, December, 1990.

Bai, S.X., and S.B. Gershwin, "Real-time production scheduling for a wafer fabrication facility," Research Report 92-19, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, 1992.

Bai, X., N. Srivatsan, and S.B. Gershwin, "Hierarchical real-time scheduling of a semiconductor fabrication facility," Proceedings, Ninth IEEE International Electronics Manufacturing Technology Symposium, Washington, D.C., October 1-3, 1990.

Baker, K.R., "Sequencing rules and due-date assignments in a job shop," *Management Science*, Vol. 30, pp. 1093-1104, 1984.

Baker, K.R., *Elements of Sequencing and Scheduling*, Amos Tuck School of Business Administration, Dartmouth College, Hanover, New Hampshire, 1992.

Baker, K.R., and J.W.M. Bertrand, "A dynamic priority rule for sequencing against due-dates," *Journal of Operations Management*, Vol. 3, pp. 37-42, 1982.

Baker, K.R., and J.J. Kanet, "Job shop scheduling with modified due dates," *Journal of Operations Management*, Vol. 4, pp. 11-22, 1983.

Barnes, J.W., and J.B. Chambers, "Solving the job shop scheduling problem using tabu search," Technical Report OPR91-06, Graduate Program in Operations Research, University of Texas at Austin, 1991.

Barnes, J.W., and M. Laguna, "Solving the multiple-machine weighted flow time problem using tabu search," Technical Report Series, Graduate Program in Operations Research, University of Texas, Austin, 1992.

Bean, J.C., "Genetics and random keys for sequencing and optimization," Technical Report 92-43, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan, 1992.

Bensana, E., G. Bel, and D. Dubois, "OPAL: a multi-knowledge-based system for industrial job-shop scheduling," *International Journal of Production Research*, Vol. 26, No. 5, pp. 795-819, 1988.

Bhaskaran, K., and M. Pinedo, "Dispatching," Chapter 83, *Handbook of Industrial Engineering*, G. Salvendy, ed., J. Wiley, New York, 1991.

Bianco, L., and S. Ricciardelli, "Scheduling of a single machine to minimize total weighted completion time subject to release dates," *Naval Research Logistics Quarterly*, Vol. 29, No. 1, pp. 151-161, 1982.

Biegel, J.E., and J.J. Davern, "Genetic algorithms and job shop scheduling," *Computers and Industrial Engineering*, 19, pp. 81-91, 1990.

Bitran, G.R., and D. Tirupati, "Planning and scheduling for epitaxial wafer production facilities," *Operations Research*, Vol. 36, pp. 34-49, 1988a.

Bitran, G.R., and D. Tirupati, "Development and implementation of a scheduling system for a wafer fabrication facility," *Operations Research*, Vol. 36, pp. 377-395, 1988b.

Blackstone, J.H., Jr., D.T. Phillips, and G.L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *International Journal of Productions Research*, Vol. 20, pp. 27-45, 1982.

Bruno, J., and P. Downey, "Complexity of task sequencing with deadlines, set-up times and changeover costs," *SIAM Journal of Computing*, Vol. 7, No. 4, pp. 393-404, 1978.

Buffa, E. S., *Production Inventory Systems: Planning and Control*, Richard D. Irwin. Inc., Homewood, Illinois, 1968.

Burman, D.Y., F.J. Gurrola-Gal, A. Nozari, S. Sathaye, and J.P. Sitarik, "Performance analysis techniques for IC manufacturing lines," *AT&T Technical Journal*, Vol. 65, pp. 46-57, 1986.

Burns, F., and J. Rooker, "Extensions and comments regarding special cases of the three machine flow-shop problem," *Naval Research Logistics Quarterly*, Vol. 22, pp. 811-817, 1975.

Cerny, V., "Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, Vol. 45, p. 41, 1985.

Chambers, R.J., R.L. Carraway, T.J. Lowe, and T.L. Morris, "Dominance and decomposition heuristics for single machine scheduling," *Operations Research*, Vol. 39, No. 4, pp. 639-647, 1991.

Chen, H., J.M. Harrison, A. Mandelbaum, A. Van Ackere, and L.M. Wein, "Empirical evaluation of a queueing network model for semiconductor wafer fabrication," *Operations Research*, Vol. 36, pp. 202-215, 1988.

Cleveland, G., and S. Smith, "Using genetic algorithms to schedule flow shop releases," in *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer, ed., Morgan Kaufmann, San Mateo, California, 1989.

Coffman, E.G., A. Nozari, and M. Yannakakis, "Optimal scheduling of products with two subassemblies on a single machine," *Operations Research*, Vol. 37, No. 3, pp. 426-436, May-June, 1989.

Conway, R.W., "Priority dispatching and job lateness in a job shop," *Journal of Industrial Engineering*, Vol. 16, pp. 228-237, 1965.

Conway, R.W., W.L. Maxwell, and L.W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, Massachusetts, 1967.

Corbett, C.J., and L.N. Van Wassenhove, "The natural drift: what happened to operations research?" *Operations Research*, Vol. 41, No. 4, pp. 625-640, 1993.

Davis, L., "Job shop scheduling with genetic algorithms," in *Proceedings of an International Conference on Genetic Algorithms and their Applications*, J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1985.

Davis, L., ed., *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London, 1987.

Davis, L., ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

Day, J. E. and M. P. Hottenstein, "Review of sequencing research," *Naval Research Logistic Quarterly*, Vol. 17, pp. 11-39, 1970.

Dayhoff, J.E., and R.W. Atherton, "Signature analysis: simulation of inventory, cycle time, and throughput trade-offs in wafer fabrication," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, Vol. 9, pp. 498-507, 1986a.

Dayhoff, J.E., and R.W. Atherton, "Signature analysis of dispatch schemes in wafer fabrication," *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, Vol. 9, pp. 518-525, 1986b.

Dayhoff, J.E., and R.W. Atherton, "A model for wafer fabrication dynamics in integrated circuit manufacturing," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 17, pp. 91-100, 1987.

Dessouky, M.I., and J.S. Deogun, "Sequencing jobs with unequal ready times to minimize mean flow time," *SIAM Journal on Computing*, Vol. 10, No. 1, pp. 192-202, 1981.

Dobson, G., U.S. Karmarkar, and J.L. Rummel, "Batching to minimize flow times on one machine," *Management Science*, Vol. 33, No. 6, pp. 784-799, June, 1987.

Dobson, G., U.S. Karmarkar, and J.L. Rummel, "Batching to minimize flow times on parallel heterogeneous machines," *Management Science*, Vol. 35, No. 5, pp. 607-613, May, 1989.

Emmons, H., "One machine sequencing to minimize mean flow time with minimum number tardy," *Naval Research Logistics Quarterly*, Vol. 22, pp. 585-592, 1975.

Faaland, B., and T. Schmitt, "Cost-based scheduling of workers and equipment in a fabrication and assembly shop," *Operations Research*, Vol. 41, No. 2, pp. 253-268, 1993.

Fisher, H., and G.L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Industrial Scheduling*, J.F. Muth and G.L. Thompson, eds., Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1963.

Fordyce, K., R. Dunki-Jacobs, B. Gerard, R. Sell, and G. Sullivan, "Logistics Management System: an advanced decision suport system for the fourth decision tier dispatch or short-interval scheduling," *Production and Operations Management*, Vol. 1, pp. 70-85, 1992.

Fox, B. R., and M. B. McMahon, "Genetic operators for sequencing problems," Planning and Scheduling Group, McDonnell Douglas Space Systems, Houston, Texas, 1990.

Fox, M.S., and S.F. Smith, "ISIS - a knowledge-based system for factory scheduling," *Expert Systems*, Vol. 1, No. 1, pp. 25-49, 1984.

Fry, T.D., P.R. Philipoom, and J.H. Blackstone, "A simulation study of processing time dispatching rules," *Journal of Operations Management*, Vol. 7, pp. 77-92, 1988.

Garey, M.R., and D.S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

Garey, M. R., D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, Vol. 1, pp. 117-129, 1976.

Gazmuri, P.G., "Probabilistic analysis of a machine scheduling problem," *Mathematics of Operations Research*, Vol. 10, No. 2, pp. 328-339, 1985.

Glassey, C.R., and R.G. Petrakian, "The use of bottleneck starvation avoidance with queue predictions in shop floor control," Research Report ESRC 89-23, University of California, Berkeley, 1989.

Glassey, C.R., and M.G.C. Resende, "Closed-loop job release control for vlsi circuit manufacturing," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 1, pp. 36-46, 1988a.

Glassey, C.R., and M.G.C. Resende, "A scheduling rule for job release in semiconductor fabrication," *Operations Research Letters*, Vol. 7, pp. 213-217, 1988b.

Glover, F., "Heuristics for integer programming using surrogate constraints," *Decision Sciences*, Vol. 8, No. 1, pp. 156-166, 1977.

Glover, F., "Tabu Search - Part I," *ORSA Journal on Computing*, Vol. 1, pp. 190-206, 1989.

Glover, F., "Tabu Search - Part II," *ORSA Journal on Computing*, Vol. 2, pp. 4-32, 1990.

Glover, F., E. Taillard, and D. de Werra, "A user's guide to tabu search," Graduate School of Business, University of Colorado, Boulder, Colorado, 1991.

Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts, 1989.

Golovin, J.J., "A total framework for semiconductor production planning and scheduling," *Solid State Technology*, pp. 167-170, May, 1986.

Gong, L., and H. Matsuo, "Stabilizing work-in-process and smoothing production in a production system with random yield," Graduate School of Business, University of Texas, Austin, May, 1990a.

Gong, L., and H. Matsuo, "A control policy for a manufacturing system with random yield and rework," Graduate School of Business, University of Texas, Austin, September, 1990b.

Gonzalez, T., and S. Sahni, "Flowshop and jobshop schedules: complexity and approximation," *Operations Research*, Vol. 26, pp. 36-52, 1978.

Grabowski, J., E. Skubalska, and C. Smutnicki, "On flow shop scheduling with release and due dates to minimize maximum lateness," *Journal of the Operational Research Society*, Vol. 34, pp. 615-620, 1983.

Graves, S.C., H.C. Meal, D. Stefek, and A.H. Zeghmi, "Scheduling of re-entrant flow shops," *Journal of Operations Management*, Vol. 3, pp. 197-207, 1983.

Green, G.I., and L.B. Appel, "An empirical analysis of job shop dispatch rule selection," *Journal of Operations Management*, Vol. 1, pp. 197-203, 1981.

Gupta, J.N.D., "Optimal schedules for single facility with classes," *Computers and Operations Research*, Vol. 11, pp. 409-413, 1984.

Gupta, J.N.D., "Single facility scheduling with multiple job classes," *European Journal of Operational Research*, Vol. 8, pp. 42-45, 1988a.

Gupta, J.N.D., "Two-stage, hybrid flowshop scheduling problem," *Journal of the Operational Research Society*, Vol. 39, No. 4, pp. 359-364, 1988b.

Gupta, J.N.D., and E.A. Tunc, "Schedule for a two-stage hybrid flowshop with parallel machines at the second stage," *International Journal of Production Research*, Vol. 29, No. 7, pp. 1489-1502, 1991.

Hadavi, K., and K. Voigt, "An integrated planning and scheduling environment," Proceedings Simulation and Artificial Intelligence in Manufacturing, Society of Manufacturing Engineers, Long Beach, California, October 14-16, 1987.

Hadavi, K., W.L. Hsu, T. Chen, and C.N. Lee, "An architecture for real time distributed scheduling," working paper, Siemens Corporate Research, 1991.

Hariri, A. M. A., and C. N. Potts, "An algorithm for single machine sequencing with release dates to minimize total weighted completion time," *Discrete Applied Mathematics*, Vol. 5, pp. 99-109, 1983.

Hariri, A. M. A., and C. N. Potts, "A branch and bound algorithm to minimize the number of late jobs in a permutation flow-shop," *European Journal of Operational Research*, 38, pp. 228-237, 1989.

Harrison, J.M., C.A. Holloway, and J.M. Patell, "Measuring delivery performance: a case study from the semiconductor industry," presented at the "Measuring Manufacturing Performance" colloquium, Harvard Business School, Cambridge, Massachusetts, January 25-26, 1989.

Herrmann, J.W., C.-Y. Lee, and J.L. Snowdon, "A classification of static scheduling problems," in *Complexity in Numerical Optimization*, P.M. Pardalos, ed., pp. 203-253, World Scientific, River Edge, New Jersey, 1993.

Ho, J.C., "Minimizing the number of tardy jobs with two job classes," Division of Business, Northeast Missouri State University, 1992.

Holland, J.H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan, 1975.

Holloway, C.A., and R.T. Nelson, "Job shop scheduling with due dates and variable processing times," *Management Science*, Vol. 20, pp. 1264-1275, 1974.

Ignall, E., and L. Schrage, "Application of the branch and bound technique to some flow-shop scheduling problems," *Operations Research*, Vol. 15, pp. 400-412, 1965.

Jackson, J.R., "Scheduling a production line to minimize maximum tardiness," Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

Jackson, J.R., "Networks on waiting lines," *Operations Research*, Vol. 5, 1967.

Jacobs, F.R., "OPT uncovered: concepts behind the system," *Industrial Engineering*, Vol. 16, No. 10, pp. 32-41, 1984

Johnson, S.M., "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, Vol. 1, pp. 61-68, 1954.

Kanet, J.J., and J.C. Hayya, "Priority dispatching with operation due-dates in a job shop," *Journal of Operations Management*, Vol. 2, pp. 155-163, 1982.

Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," *Science*, Vol. 220, p. 671, 1983.

Kise, H., T. Ibaraki, and H. Mine, "A solvable case of the one-machine scheduling problem with ready and due times", *Operations Research*, Vol. 26, pp. 121-126, 1978.

Kohler, W.H., and K. Steiglitz, "Exact, approximate, and guaranteed accuracy algorithms for the flow-shop problem n/2/F/Favg," *Journal of the Association for Computing Machinery*, Vol. 22, pp. 106-114, 1975.

Koulamas, C.P., and M.L. Smith, "Look-ahead scheduling for minimizing machine interference," *International Journal of Productions Research*, Vol. 26, pp. 1523-1533, 1988.

Krone, M. J., and K. Steiglitz, "Heuristic-programming solution of a flowshop-scheduling problem," *Operations Research*, Vol. 22, pp. 629-638, 1974.

Kubiak, W., S.X.C. Lou, and Y.M. Wang, "Mean flow time minimization in re-entrant job shops with hub," Faculty of Management, University of Toronto, July 17, 1990.

van Laarhoven, P.J.M., E.H.L. Aarts, and J.K. Lenstra, "Job shop scheduling by simulated annealing," Report OS-R8809, Centre for Mathematics and Computer Science, 1988.

Lageweg, B. J., J. K. Lenstra, and A. H. G. Rinnooy Kan, "A general bounding scheme for the permutation flow-shop problem," *Operations Research*, Vol. 26, No. 1, pp. 53-67, 1978.

Laguna, M., J.W. Barnes, and F. Glover, "Tabu search methods for a single machine scheduling problem," Technical Report Series, Graduate Program in Operations Research, University of Texas, Austin, 1989.

Laguna M., and F. Glover, "Integrating target analysis and tabu search for improved scheduling systems," Graduate School of Business, University of Colorado, Boulder, Colorado, 1991.

Lawler, E.L., "Scheduling a single machine to minimize the number of late jobs," Preprint, Computer Science Division, University of California, Berkeley, 1982.

Leachman, R.C., "Preliminary design and development of a corporate-level production planning system for the semiconductor industry," Operations Research Center, University of California, Berkeley, February, 1986.

Leachman, R.C., M. Solorzano, and C.R. Glassey, "A queue management policy for the release of factory work orders," Research Report 88-19, Engineering Systems Research Center, University of California at Berkeley, 1988.

Leachman, R.C., and V.S. Sohoni, "Automated shift scheduling as a tool for problem identification and people management in semiconductor factories," University of California, Berkeley, California.

Lee, C.-Y., T.C.E. Cheng, and B.M.T. Lin, "Exact and approximate solutions to the 3-machine assembly-type flowshop scheduling problem to minimize the makespan," *Management Science*, Vol. 39, No. 5, pp. 616-625, 1993.

Lee, C.-Y., and J.W. Herrmann, "A three-machine scheduling problem with look-behind characteristics," Research Report 93-11, Department of Industrial and Systems Engineering, University of Florida, Gainesville, Florida, 1993.

Lee, C.-Y., L.A. Martin-Vega, R. Uzsoy, and J. Hinchman, "Implementation of a decision support system for scheduling semiconductor testing operations," to appear in *Journal of Electronic Manufacturing*, 1993.

Lee, C.-Y., R. Uzsoy, and L.A. Martin-Vega, "Efficient algorithms for scheduling semiconductor burn-in operations," *Operations Research*, Vol. 40, No. 4, pp. 764-775, 1992.

Lenstra, J.K., A.H.G. Rinnooy Kan, and P. Brucker, "Complexity of machine scheduling problems," *Annals of Discrete Mathematics*, Vol. 1, pp. 343-362, 1977.

Liepins, G.E., and M.R. Hilliard, "Genetic algorithms: foundations and applications," *Annals of Operations Research*, Vol. 21, p. 31-58, 1989.

Liu, J., and B.L. MacCarthy, "Effective heuristics for the single machine sequencing problem with ready times," *International Journal of Production Research*, Vol. 29, No. 8, pp. 1521-1533, 1991.

Lou, S.X.C., and P.W. Kager, "A robust control policy for vlsi wafer fabrication," *IEEE Transactions on Semiconductor Manufacturing*, Vol. 2, pp. 159-164, 1989.

Lundrigan, R., "What is this thing they call OPT?" *Production and Inventory Management*, Vol. 27, pp. 2-12, 1986.

Malek, M., M. Guruswamy, M. Pandya, H. Owens, "Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem," *Annals of Operations Research*, Vol. 21, p. 59-84, 1989.

Mason, A. J., and E. J. Anderson, "Minimizing flow time on a single machine with job classes and setup times," *Naval Research Logistics*, Vol. 38, pp. 333-350, 1991.

Masuda, T., H. Ishii, and T. Nishida, "Some bounds on approximation algorithms for n/m/I/Lmax and n/2/F/Lmax scheduling problems," *Journal of the Operations Research Society of Japan*, Vol. 26, pp. 212-224, 1983.

Matsuo, H., C.J. Suh, and R.S. Sullivan, "Controlled search simulated annealing for general job shop scheduling problem," Working Paper #03-04-88, 1988.

Matsuo, H., C.J. Suh, and R.S. Sullivan, "A controlled search simulated annealing method for the single-machine weighted tardiness problem," *Annals of Operations Research*, Vol. 21, p. 85-108, 1989.

Meleton, M.P., "OPT - fantasy or breakthrough," *Production and Inventory Management*, Vol. 27, pp. 13-21, 1986.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, Vol. 21, pp. 1087-1092, 1953.

Mitten, L.G., "Sequencing *n* jobs on two machines with arbitrary time lags," *Management Science*, Vol. 5, pp. 293-298, 1958.

Miyazaki, S., and N. Nishiyama, "Analysis of minimizing weighted mean flow-time in flow-shop scheduling," *Journal of the Operations Research Society of Japan*, Vol. 23, pp. 118-132, 1980.

Miyazaki, S., N. Nishiyama, and F. Hashimoto, "An adjacent pairwise approach to the mean flow-time scheduling problem," *Journal of the Operations Research Society of Japan*, Vol. 21, pp. 287-299, 1978.

Monma, C. L., and C. N. Potts, "On the complexity of scheduling with batch setup times," *Operations Research*, Vol. 37, No. 5, pp. 798-804, September-October, 1989.

Moore, J.M., "An *n* job, one machine sequencing algorithm for minimizing the number of late jobs," *Management Science*, Vol. 15, pp. 102-109, 1968.

Morton, T.E., *Heuristic Scheduling Systems*, Graduate School of Industrial Administration, Carnegie Mellon University, John Wiley, in press, 1992.

Morton, T.E., S.R. Lawrence, S. Rajagopolan, and S. Kekre, "SCHED-STAR: a price-based shop scheduling module," *Journal of Manufacturing and Operations Management*, Vol. 1, pp. 131-181, 1988.

Morton, T.E., and P. Ramnath, "Guided forward tabu/beam search for scheduling very large dynamic job shops. I: one machine weighted tardiness," Working paper 1992-47, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992.

Najmi, A., and C. Lozinski, "Managing factory productivity using object-oriented simulation for setting shift production targets in VLSI manufacturing," Proceedings of the Autofact Conference, Society of Manufacturing Engineers, Detroit, Michigan, pp. 3.1-3.14, November, 1989.

Nakano, R., and T. Yamada, "Conventional genetic algorithms for job shop problems," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R.K. Below and L.B. Booker, eds., Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

Ogbu, F.A., and D.K. Smith, "The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem," *Computers and Operations Research*, Vol. 17, p. 243, 1990.

Oliver, I.M., D.J. Smith, and J.R.C. Holland, "A study of permutation crossover operators on the traveling salesman problem," *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, J. Grefenstette, ed., Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987.

Ou, J., and L.M. Wein, "Dynamic scheduling of a production/inventory system with by-products and random yield," Sloan School of Management, Massachusetts Institute of Technology, May, 1991.

Ow, P.S., and S.F. Smith, "Viewing scheduling as an opportunistic problem-solving process," *Annals of Operations Research*, Vol. 12, pp. 85-108, 1988.

Panwalker, S.S., and W. Iskander, "A survey of scheduling rules," *Operations Research*, Vol. 25, pp. 45-61, 1977.

Posner, M.E., "Minimizing weighted completion times with deadlines," *Operations Research*, Vol. 33, pp. 562-274, 1985.

Potts, C. N., "Scheduling two job classes on a single machine," *Computers and Operations Research*, Vol. 18, pp. 411-415, 1991.

Potts, C.N., and L.N. van Wassenhove, "An algorithm for single machine sequencing with deadlines to minimize total weighted completion time," *European Journal of Operational Research*, Vol. 12, pp. 379-387, 1983.

Rachamadugu, R.M., and T.E. Morton, "Myopic heuristics for the single machine weighted tardiness problem," Working Paper 30-82-83, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1982.

Rao, H.R., and B.P. Lingaraj, "Expert systems in production and operations management: classification and prospects," *Interfaces*, Vol. 18, No. 6, pp. 80-91, 1988.

Reeves, C.R., "Improving the efficiency of tabu search for machine sequencing problems," *Journal of the Operational Research Society*, Vol. 44, No. 4, pp. 375-382, 1993.

Rinaldi, G., and A. Sassano, "On a job scheduling problem with different ready times: some properties and a new algorithm to determine the optimal solution," Report R.77-24, Istituto di Automatica, Universita di Roma, 1977.

Robinson, J.K., J.W. Fowler, and J.F. Bard, "The use of upstream and downstream information in scheduling semiconducotr batch operations," Technical Report 93-06, Graduate Program in Operations Research, University of Texas, Austin, 1993.

Sadeh, N., "MICRO-BOSS: a micro-opportunistic factory scheduler," Center for Integrated Manufacturing Decision Systems, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1991.

Sahney, V. K., "Single-server, two-machine sequencing with switching time," *Operations Research*, Vol. 20, pp. 24-36, 1972.

Savell, D.V., R.A. Perez, and S.W. Koh, "Scheduling semiconductor wafer production: an expert system implementation," *IEEE Expert*, Vol. 4, pp. 9-15, 1989.

Schutten, J.M.J., and W.H.M. Zijm, "Scheduling a single machine with release dates, due dates, and family setup times," Department of Mechanical Engineering, University of Twente, The Netherlands, 1993.

Shanthikumar, J.G., "Scheduling jobs on one machine to minimize the maximum tardiness with minimum number tardy," *Journal of Computers and Operations Research*, Vol. 10, pp. 255-266, 1983.

Smith, S.F., M.S. Fox, and P.S. Ow, "Constructing and maintaining detailed production plans: investigations into the development of knowledge-based factory scheduling systems," *AI Magazine*, Vol. 7, No. 4, pp. 45-61, 1986.

Smith, W.E., "Various optimizers for single-stage production," *Naval Research Logistics Quarterly*, Vol. 3, pp. 59-66, 1956.

Solorzano, M., "Workload Regulation of Semiconductor Fabrication Facilities," ESRC 89-1, University of California, Berkeley, January, 1989.

Spence, A.M., and D.J. Welter, "Capacity Planning of a Photolithography Work Cell in a Wafer Manufacturing Line," Proceedings of the IEEE Conference on Robotics and Automation, pp. 702-708, 1987.

Sriskandarajah, C., and S.P. Sethi, "Scheduling algorithms for flexible flowshops: worst and average case performance," *European Journal of Operational Research*, Vol. 43, pp. 143-160, 1989.

Starkweather, T., S. McDaniels, K. Mathias, C. Whitley, and D. Whitley, "A comparison of genetic sequencing operators," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, R.K. Below and L.B. Booker, eds., Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

Storer, R.H., S.Y.D. Wu, and R. Vaccari, "Local Search in Problem and Heuristic Space For Job Shop Scheduling," Working Paper, Department of Industrial Engineering, Lehigh University, 1990.

Storer, R.H., S.Y.D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management Science*, Vol. 38, No. 10, pp. 1495-1509, 1992.

Sullivan, G., and K. Fordyce, "IBM Burlington's Logistics Management System," *Interfaces*, Vol. 20, pp. 43-64, 1990.

Syswerda, G., "Uniform crossover in genetic algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer, ed., Morgan Kaufmann Publishers, Inc., San Mateo, California, 1989.

Syswerda, G., "Schedule optimization using genetic algorithms," in *Handbook of Genetic Algorithms*, L. Davis, ed., Van Nostrand Reinhold, 1991.

Szwarc, W., "Optimal two-machine orderings in the 3 × *n* flow-shop problem," *Operations Research*, Vol. 25, pp. 70-77, 1977.

Szwarc, W., "The flow-shop problem with mean completion time criterion," *IIE Transactions*, Vol. 15, pp. 172-176, 1983.

Trilling, D. R., "Job shop simulation of orders that are networks," *Journal of Industrial Engineering*, Vol. 17, 1966.

Uzsoy, R., C.-Y. Lee, and L.A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry, part I: system characteristics, performance evaluation and production planning," *IIE Transactions*, Vol. 24, pp. 47-61, 1992a.

Uzsoy, R., C.-Y. Lee, and L.A. Martin-Vega, "Scheduling semiconductor test operations: minimizing maximum lateness and number of tardy jobs on a single machine," *Naval Research Logistics*, Vol. 39, pp. 369-388, 1992b.

Uzsoy, R., C.-Y. Lee, and L.A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry, part II: shop floor control," to appear in *IIE Transactions*, 1993.

Uzsoy, R., C.-Y. Lee, L.A. Martin-Vega, and J. Hinchman, "Scheduling Semiconductor Testing Operations: Optimization and Approximation," Proceedings, Joint U.S.-German Conference on New Directions for Operations Research in Manufacturing, Gaithersburg, Maryland, July 30-31, 1991a.

Uzsoy, R., L.A. Martin-Vega, C.-Y. Lee, and P.A. Leonard, "Production Scheduling Algorithms for a Semiconductor Test Facility," *IEEE Transactions on Semiconductor Manufacturing*, 1991b.

Vairaktarakis, G.L., and C.-Y. Lee, "The single machine problem to minimize total tardiness subject to minimum number of tardy jobs," to appear in *IIE Transactions*, 1993.

Vakharia, A.J., and Y.L. Chang, "A simulated annealing approach to scheduling a manufacturing cell," *Naval Research Logistics*, Vol. 37, p. 559, 1990.

Van de Velde, S.L., "Minimizing the sum of the job completion times in the two-mchine flow shop by Lagrangian relaxation," *Annals of Operations Research*, Vol. 26, pp. 257-268, 1990.

Vepsalainen, A.P.J., and T.E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management Science*, Vol. 33, No. 8, pp. 1035-1047, 1987.

Vepsalainen, A.P.J., and T.E. Morton, "Improving local priority rules with global lead-time estimates," *Journal of Manufacturing and Operations Management*, Vol. 1, pp. 102-118, 1988.

Vollmann, T.E., "OPT as an enhancement to MRP II," *Production and Inventory Management*, Vol. 27, pp. 38-47, 1986.

Vollmann, T.E., W.L. Berry, and D.C. Whybark, *Manufacturing Planning and Control Systems*, Second edition, Dow Jones-Irwin, Homewood, Illinois, 1988.

Wein, L.M., "Scheduling semiconductor wafer fabrication" *IEEE Transactions on Semiconductor Manufacturing*, Vol. 1, pp. 115-130, 1988.

Wein, L.M., "On the relationship between yield and cycle time in semiconductor wafer fabrication," Sloan School of Management, Massachusetts Institute of Technology, January, 1991.

Wein, L.M., and P.B. Chevalier, "A broader view of the job-shop scheduling problem," *Management Science*, Vol. 38, No. 7, pp. 1018-1029, 1992.

Whitley, D., "GENITOR: a different genetic algorithm," in *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver, Colorado, 1988.

Whitley, D., T. Starkweather, and D. Shaner, "The traveling salesman and sequence scheduling: quality solutions using genetic edge optimization," in *Handbook of Genetic Algorithms*, L. Davis, ed., Van Nostrand Reinhold, 1991.

Widmer, M., and A. Hertz, "A new heuristic method for the flow shop sequencing problem," *European Journal of Operational Research*, Vol. 41, p. 186-193, 1989.

Wittrock, R.J., "An adaptable scheduling algorithm for flexible flow lines," *Operations Research*, Vol. 36, No. 3, pp. 445-453, 1988.

Woodruff, D. L., and M. L. Spearman, "Sequencing and batching for two classes of jobs with deadlines and setup times," *Production and Operations Management*, Vol. 1, pp. 87-102, 1992.

Wu, S.D., R.H. Storer, and P.-C. Chang, "One-machine rescheduling heruistics with efficiency and stability as criteria," *Computers and Operations Research*, Vol. 20, No. 1, pp. 1-14, 1993.

Zeestraten, M. J., "The look ahead dispatching procedure," *International Journal of Production Research*, Vol. 28, pp. 369-384, 1990.

BIOGRAPHICAL SKETCH

Jeffrey W. Herrmann has a degree in applied mathematics from Georgia Tech and is a Ph.D. candidate in the Department of Industrial and Systems Engineering at the University of Florida. He held an NSF Graduate Research Fellowship from 1990 until 1993 and has been working on an applied research project with Harris Semiconductor. His research interests include operations research, production scheduling, and heuristic search.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Chung-Yee Lee, Chairman
Associate Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jack Elzinga

Jack Elzinga
Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sherman Bai
Assistant Professor of Industrial and Systems Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Gary Koehler
Professor of Decision and Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Selcuk Erenguc
Professor of Decision and Information Sciences

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

December, 1993

for Winfred M. Phillips
Dean, College of Engineering

Karen Holbrook
Dean, Graduate School

MARSTON SCIENCE LIBRARY

Date Due

Due	Returned	Due	Returned
~~MAY 3 0 1995~~			
MAY 2 9 1995	MAY 2 4 1995		
JAN 1 5 1996	DEC 1 3 1996		
JAN 0 6 1997	NOV 1 8 1996		
MAR 1 7 1997	MAR 2 4 1997		
[illegible]			
[illegible]			